User Guide

AT32F435/437 Firmware BSP&Pack

Introduction

This application note is written to give a quick guideline on how to apply AT32F435/437 BSP (Board Support Package) and install AT32 Pack.

Contents

1 Overview 48

2 How to install Pack 49

2.1 IAR Pack installation 49

2.2 Keil_v5 Pack installation 51

2.3 Keil_v4 Pack installation 52

2.4 Segger Pack installation 55

3 Flash algorithm file 59

3.1 How to use Keil algorithm file 59

3.2 How to use IAR algorithm files 61

4 BSP introduction 65

4.1 Quick start 65

4.1.1 Template project 65

4.1.2 BSP macro definitions 66

4.2 BSP specifications 68

4.2.1 List of abbreviations for peripherals 68

4.2.2 Naming rules 69

4.2.3 Encoding rules 70

4.3 BSP structure 73

4.3.1 BSP folder structure 73

4.3.2 BSP function library structure 74

4.3.3 Initialization and configuration for peripherals 75

4.3.4 Peripheral functions format description 75

5 AT32F435/437 peripheral library functions 76

5.1 HICK automatic clock calibration (ACC) 76

5.1.1 acc_calibration_mode_enable function 77

5.1.2 acc_step_set function 77

5.1.3 acc_interrupt_enable function 78

5.1.4 acc_hicktrim_get function 79

5.1.5 acc_hickcal_get function 79
5.1.6 acc_write_c1 function 80
5.1.7 acc_write_c2 function 80
5.1.8 acc_write_c3 function 81
5.1.9 acc_read_c1 function 81
5.1.10 acc_read_c2 function 82
5.1.11 acc_read_c3 function 82
5.1.12 acc_flag_get function 83
5.1.13 acc_interrupt_flag_get function 83
5.1.14 acc_flag_clear function 84
5.2 Analog-to-digital converter (ADC) 84
5.2.1 adc_reset function 87
5.2.2 adc_enable function 87
5.2.3 adc_base_default_para_init function 88
5.2.4 adc_base_config function 88
5.2.5 adc_common_default_para_init function 89
5.2.6 adc_common_config function 90
5.2.7 adc_resolution_set function 93
5.2.8 adc_voltage_battery_enable function 94
5.2.9 adc_dma_mode_enable function 94
5.2.10 adc_dma_request_repeat_enable function 95
5.2.11 adc_interrupt_enable function 95
5.2.12 adc_calibration_value_set 96
5.2.13 adc_calibration_init function 96
5.2.14 adc_calibration_init_status_get function 97
5.2.15 adc_calibration_start function 97
5.2.16 adc_calibration_status_get function 98
5.2.17 adc_voltage_monitor_enable function 98
5.2.18 adc_voltage_monitor_threshold_value_set function 99
5.2.19 adc_voltage_monitor_single_channel_select function 100
5.2.20 adc_ordinary_channel_set function 100
5.2.21 adc_preempt_channel_length_set function 101
5.2.22 adc_preempt_channel_set function 102
5.2.23 adc_ordinary_conversion_trigger_set function 102
5.2.24 adc_preempt_conversion_trigger_set function 104

5.2.25 adc_preempt_offset_value_set function 105
5.2.26 adc_ordinary_part_count_set function 106
5.2.27 adc_ordinary_part_mode_enable function 106
5.2.28 adc_preempt_part_mode_enable function 107
5.2.29 adc_preempt_auto_mode_enable function 107
5.2.30 adc_conversion_stop function 108
5.2.31 adc_conversion_stop_status_get function 108
5.2.32 adc_occe_each_conversion_enable function 109
5.2.33 adc_ordinary_software_trigger_enable function 109
5.2.34 adc_ordinary_software_trigger_status_get function 110
5.2.35 adc_preempt_software_trigger_enable function 110
5.2.36 adc_preempt_software_trigger_status_get function 111
5.2.37 adc_ordinary_conversion_data_get function 111
5.2.38 adc_combine_ordinary_conversion_data_get function 112
5.2.39 adc_preempt_conversion_data_get function 112
5.2.40 adc_flag_get function 113
5.2.41 adc_interrupt_flag_get function 113
5.2.42 adc_flag_clear function 114
5.2.43 adc_ordinary_oversample_enable function 114
5.2.44 adc_preempt_oversample_enable function 115
5.2.45 adc_oversample_ratio_shift_set function 115
5.2.46 adc_ordinary_oversample_trig_enable function 116
5.2.47 adc_ordinary_oversample_restart_set function 117
5.3 Controller area network (CAN) 118
5.3.1 can_reset function 120
5.3.2 can_baudrate_default_para_init function 120
5.3.3 can_baudrate_set function 121
5.3.4 can_default_para_init function 122
5.3.5 can_base_init function 122
5.3.6 can_filter_default_para_init function 124
5.3.7 can_filter_init function 124
5.3.8 can_debug_transmission_prohibit function 126
5.3.9 can_ttc_mode_enable function 126
5.3.10 can_message_transmit function 127
5.3.11 can_transmit_status_get function 128

5.3.12 can_transmit_cancel function 129
5.3.13 can_message_receive function 130
5.3.14 can_receive_fifo_release function 131
5.3.15 can_receive_message_pending_get function 132
5.3.16 can_operating_mode_set function 132
5.3.17 can_doze_mode_enter function 133
5.3.18 can_doze_mode_exit function 133
5.3.19 can_error_type_record_get function 134
5.3.20 can_receive_error_counter_get function 135
5.3.21 can_transmit_error_counter_get function 135
5.3.22 can_interrupt_enable 136
5.3.23 can_flag_get function 137
5.3.24 can_interrupt_flag_get function 138
5.3.25 can_flag_clear function 139
5.4 CRC calculation unit (CRC) 140
5.4.1 crc_data_reset function 141
5.4.2 crc_one_word_calculate function 141
5.4.3 crc_block_calculate function 142
5.4.4 crc_data_get function 142
5.4.5 crc_common_data_set function 143
5.4.6 crc_common_data_get function 143
5.4.7 crc_init_data_set function 144
5.4.8 crc_reverse_input_data_set function 144
5.4.9 crc_reverse_output_data_set function 145
5.4.10 crc_poly_value _set function 145
5.4.11 crc_poly_value _get function 146
5.4.12 crc_poly_size _set function 146
5.4.13 crc_poly_size _get function 147
5.5 Clock and reset management (CRM) 148
5.5.1 crm_reset function 150
5.5.2 crm_lext_bypass function 150
5.5.3 crm_hext_bypass function 151
5.5.4 crm_flag_get function 151
5.5.5 crm_interrupt_flag_get function 152
5.5.6 crm_hext_stable_wait function 153

5.5.7 crm_hick_clock_trimming_set function 153
5.5.8 crm_hick_clock_calibration_set function 154
5.5.9 crm_periph_clock_enable function 154
5.5.10 crm_periph_reset function 155
5.5.11 crm_periph_lowpower_mode_enable function 156
5.5.12 crm_clock_source_enable function 156
5.5.13 crm_flag_clear function 157
5.5.14 crm_ertc_clock_select function 158
5.5.15 crm_ertc_clock_enable function 158
5.5.16 crm_ahb_div_set function 159
5.5.17 crm_apb1_div_set function 159
5.5.18 crm_apb2_div_set function 160
5.5.19 crm_usb_clock_div_set function 160
5.5.20 crm_clock_failure_detection_enable function 161
5.5.21 crm_battery_powered_domain_reset function 161
5.5.22 crm_pll_config function 162
5.5.23 crm_sysclk_switch function 163
5.5.24 crm_sysclk_switch_status_get function 163
5.5.25 crm_clocks_freq_get function 164
5.5.26 crm_clock_out1_set function 165
5.5.27 crm_clock_out2_set function 165
5.5.28 crm_clkout_div_set function 166
5.5.29 crm_interrupt_enable function 167
5.5.30 crm_auto_step_mode_enable function 167
5.5.31 crm_hick_sclk_frequency_select function 168
5.5.32 crm_usb_clock_source_select function 168
5.5.33 crm_clkout_to_tmr10_enable function 169
5.5.34 crm_emac_output_pulse_set function 169
5.5.35 crm_pll_parameter_calculate function 170
5.6 Digital-to-analog converter (DAC) 171
5.6.1 dac_reset function 172
5.6.2 dac_enable function 172
5.6.3 dac_output_buffer_enable function 173
5.6.4 dac_trigger_enable function 173
5.6.5 dac_trigger_select function 174

5.6.6 dac_software_trigger_generate function 174
5.6.7 dac_dual_software_trigger_generate function 175
5.6.8 dac_wave_generate function 175
5.6.9 dac_mask_amplitude_select function 176
5.6.10 dac_dma_enable function 176
5.6.11 dac_data_output_get function 177
5.6.12 dac_1_data_set function 177
5.6.13 dac_2_data_set function 178
5.6.14 dac_dual_data_set function 179
5.6.15 dac_udr_enable function 180
5.6.16 dac_udr_flag_get function 180
5.6.17 dac_udr_interrupt_flag_get function 181
5.6.18 dac_udr_flag_clear function 181
5.7 Debug 182
5.7.1 debug_device_id_get function 183
5.7.2 debug_periph_mode_set function 183
5.8 DMA controller 185
5.8.1 dma_default_para_init function 187
5.8.2 dma_init function 188
5.8.3 dma_reset function 190
5.8.4 dma_data_number_set function 190
5.8.5 dma_data_number_get function 191
5.8.6 dma_interrupt_enable function 191
5.8.7 dma_channel_enable function 192
5.8.8 dma_flag_get function 192
5.8.9 dma_flag_clear function 194
5.8.10 dma_flexible_config function 196
5.8.11 dmamux_enable function 198
5.8.12 dmamux_init function 199
5.8.13 dmamux_sync_default_para_init function 199
5.8.14 dmamux_sync_config function 200
5.8.15 dmamux_generator_default_para_init function 202
5.8.16 dmamux_generator_config function 202
5.8.17 dmamux_sync_interrupt_enable function 204
5.8.18 dmamux_generator_interrupt_enable function 205

5.8.19 dmamux_sync_flag_get function 205
5.8.20 dmamux_sync_flag_clear function 206
5.8.21 dmamux_generator_flag_get function 207
5.8.22 dmamux_generator_flag_clear function 208
5.9 Digital video parallel interface (DVP) 209
5.9.1 dvp_reset function 210
5.9.2 dvp_capture_enable function 211
5.9.3 dvp_capture_mode_set function 211
5.9.4 dvp_window_crop_enable function 211
5.9.5 dvp_window_crop_set function 212
5.9.6 dvp_jpeg_enable function 212
5.9.7 dvp_sync_mode_set function 213
5.9.8 dvp_sync_code_set function 213
5.9.9 dvp_sync_unmask_set function 214
5.9.10 dvp_pclk_polarity_set function 214
5.9.11 dvp_hsync_polarity_set function 215
5.9.12 dvp_vsync_polarity_set function 215
5.9.13 dvp_basic_frame_rate_control_set function 216
5.9.14 dvp_pixel_data_length_set function 216
5.9.15 dvp_enable function 217
5.9.16 dvp_zoomout_select function 217
5.9.17 dvp_zoomout_set function 218
5.9.18 dvp_basic_status_get function 219
5.9.19 dvp_interrupt_enable function 219
5.9.20 dvp_interrupt_flag_get function 220
5.9.21 dvp_flag_get function 220
5.9.22 dvp_flag_clear function 221
5.9.23 dvp_enhanced_scaling_resize_enable function 222
5.9.24 dvp_enhanced_scaling_resize_set function 222
5.9.25 dvp_enhanced_framerate_set function 223
5.9.26 dvp_monochrome_image_binarization_set function 223
5.9.27 dvp_enhanced_data_format_set function 224
5.9.28 dvp_input_data_unused_set function 224
5.9.29 dvp_dma_burst_set function 225
5.9.30 dvp_sync_event_interrupt_set function 226

5.10 EDMA controller (EDMA) 227
5.10.1 edma_reset function 230
5.10.2 edma_init function 231
5.10.3 edma_default_para_init function 233
5.10.4 edma_stream_enable function 234
5.10.5 edma_interrupt_enable function 235
5.10.6 edma_peripheral_inc_offset_set function 235
5.10.7 edma_flow_controller_enable 236
5.10.8 edma_data_number_set function 237
5.10.9 edma_data_number_get function 237
5.10.10 edma_double_buffer_mode_init function 238
5.10.11 edma_double_buffer_mode_enable function 238
5.10.12 edma_memory_addr_set function 239
5.10.13 edma_stream_status_get function 239
5.10.14 edma_fifo_status_get function 240
5.10.15 edma_flag_get function 240
5.10.16 edma_2d_init function 242
5.10.17 edma_2d_enable function 243
5.10.18 edma_link_list_init function 243
5.10.19 edma_link_list_enable function 244
5.10.20 edma_flag_clear function 244
5.10.21 edmamux_enable function 245
5.10.22 edmamux_init function 246
5.10.23 edmamux_sync_default_para_init function 248
5.10.24 edmamux_sync_config function 248
5.10.25 edmamux_generator_default_para_init function 250
5.10.26 edmamux_generator_config function 251
5.10.27 edmamux_sync_interrupt_enable function 252
5.10.28 edmamux_generator_interrupt_enable function 253
5.10.29 edmamux_sync_flag_get function 253
5.10.30 edmamux_sync_flag_clear function 254
5.10.31 edmamux_generator_flag_get function 255
5.10.32 edmamux_generator_flag_clear function 256
5.11 Real-time clock (ERTC) 257
5.11.1 ertc_num_to_bcd function 259

5.11.2 ertc_bcd_to_num function 259
5.11.3 ertc_write_protect_enable function 260
5.11.4 ertc_write_protect_disable function 260
5.11.5 ertc_wait_update function 260
5.11.6 ertc_wait_flag function 261
5.11.7 ertc_init_mode_enter function 261
5.11.8 ertc_init_mode_exit function 262
5.11.9 ertc_reset function 262
5.11.10 ertc_divider_set function 263
5.11.11 ertc_hour_mode_set function 263
5.11.12 ertc_date_set function 264
5.11.13 ertc_time_set function 264
5.11.14 ertc_calendar_get function 265
5.11.15 ertc_sub_second_get function 266
5.11.16 ertc_alarm_mask_set function 266
5.11.17 ertc_alarm_week_date_select function 267
5.11.18 ertc_alarm_set function 268
5.11.19 ertc_alarm_sub_second_set function 269
5.11.20 ertc_alarm_enable function 270
5.11.21 ertc_alarm_get function 270
5.11.22 ertc_alarm_sub_second_get function 272
5.11.23 ertc_wakeup_clock_set function 272
5.11.24 ertc_wakeup_counter_set function 273
5.11.25 ertc_wakeup_counter_get function 273
5.11.26 ertc_wakeup_enable function 274
5.11.27 ertc_smooth_calibration_config function 274
5.11.28 ertc_coarse_calibration_set function 275
5.11.29 ertc_coarse_calibration_enable function 275
5.11.30 ertc_cal_output_select function 276
5.11.31 ertc_cal_output_enable function 276
5.11.32 ertc_time_adjust function 277
5.11.33 ertc_daylight_set function 278
5.11.34 ertc_daylight_bpr_get function 278
5.11.35 ertc_refer_clock_detect_enable function 279
5.11.36 ertc_direct_read_enable function 279

5.11.37 ertc_output_set function 280
5.11.38 ertc_timestamp_pin_select function 281
5.11.39 ertc_timestamp_valid_edge_set function 281
5.11.40 ertc_timestamp_enable function 282
5.11.41 ertc_timestamp_get function 282
5.11.42 ertc_timestamp_sub_second_get function 283
5.11.43 ertc_tamper_1_pin_select function 284
5.11.44 ertc_tamper_pull_up_enable function 284
5.11.45 ertc_tamper_precharge_set function 285
5.11.46 ertc_tamper_filter_set function 286
5.11.47 ertc_tamper_detect_freq_set function 286
5.11.48 ertc_tamper_valid_edge_set function 287
5.11.49 ertc_tamper_timestamp_enable function 288
5.11.50 ertc_tamper_enable function 288
5.11.51 ertc_interrupt_enable function 289
5.11.52 ertc_interrupt_get function 289
5.11.53 ertc_flag_get function 290
5.11.54 ertc_interrupt_flag_get function 291
5.11.55 ertc_flag_clear function 291
5.11.56 ertc_bpr_data_write function 292
5.11.57 ertc_bpr_data_read function 293
5.12 External interrupt/event controller (EXINT) 294
5.12.1 exint_reset function 294
5.12.2 exint_default_para_init function 295
5.12.3 exint_init function 295
5.12.4 exint_flag_clear function 296
5.12.5 exint_flag_get function 297
5.12.6 exint_interrupt_flag_get function 297
5.12.7 exint_software_interrupt_event_generate function 298
5.12.8 exint_interrupt_enable function 298
5.12.9 exint_event_enable function 299
5.13 Flash memory controller (FLASH) 299
5.13.1 flash_flag_get function 301
5.13.2 flash_flag_clear function 302
5.13.3 flash_operation_status_get function 302

5.13.4 flash_bank1_operation_status_get function 303
5.13.5 flash_bank2_operation_status_get function 303
5.13.6 flash_operation_wait_for function 304
5.13.7 flash_bank1_operation_wait_for function 304
5.13.8 flash_bank2_operation_wait_for function 305
5.13.9 flash_unlock function 305
5.13.10 flash_bank1_unlock function 306
5.13.11 flash_bank2_unlock function 306
5.13.12 flash_lock function 306
5.13.13 flash_bank1_lock function 307
5.13.14 flash_bank2_lock function 307
5.13.15 flash_sector_erase function 307
5.13.16 flash_block_erase function 308
5.13.17 flash_internal_all_erase function 308
5.13.18 flash_bank1_erase function 309
5.13.19 flash_bank2_erase function 309
5.13.20 flash_user_system_data_erase function 310
5.13.21 flash_eopb0_config function 310
5.13.22 flash_word_program function 311
5.13.23 flash_halfword_program function 311
5.13.24 flash_byte_program function 312
5.13.25 flash_user_system_data_program function 313
5.13.26 flash_epp_set function 313
5.13.27 flash_epp_status_get function 314
5.13.28 flash_fap_enable function 315
5.13.29 flash_fap_status_get function 315
5.13.30 flash_ssb_set function 316
5.13.31 flash_ssb_status_get function 317
5.13.32 flash_interrupt_enable function 317
5.13.33 flash_slib_enable function 318
5.13.34 flash_slib_disable function 318
5.13.35 flash_slib_remaining_count_get function 319
5.13.36 flash_slib_state_get function 319
5.13.37 flash_slib_start_sector_get function 320
5.13.38 flash_slib_inststart_sector_get function 320

5.13.39flash_slib_end_sector_get function....321
5.13.40flash_crc_calibrate function....321
5.13.41flash_nzw_boost_enable function....322
5.13.42flash_continue_read_enable function....322
5.14 General-purpose I/Os and multiplexed I/Os (GPIO/IOMUX)....323
5.14.1 gpio_reset function....324
5.14.2 gpio_init function....324
5.14.3 gpio_default_para_init function....326
5.14.4 gpio_input_data_bit_read function....326
5.14.5 gpio_input_data_read function....327
5.14.6 gpio_output_data_bit_read function....327
5.14.7 gpio_output_data_read function....328
5.14.8 gpio_bits_set function....328
5.14.9 gpio_bits_reset function....329
5.14.10gpio_bits_write function....329
5.14.11gpio_bits_toggle function....330
5.14.12gpio_port_write function....330
5.14.13gpio_pin_wp_config function....331
5.14.14gpio_pins_huge_driven_config function....331
5.14.15gpio_pin_mux_config function....332
5.15 I2C interface....333
5.15.1 i2c_reset function....335
5.15.2 i2c_init function....336
5.15.3 i2c_own_address1_set function....336
5.15.4 i2c_own_address2_set function....337
5.15.5 i2c_own_address2_enable function....337
5.15.6 i2c_smbus_enable function....338
5.15.7 i2c_enable function....339
5.15.8 i2c_clock_stretch_enable function....339
5.15.9 i2c_ack_enable function....340
5.15.10i2c_addr10_mode_enable function....340
5.15.11i2c_transfer_addr_set function....341
5.15.12i2c_transfer_addr_get function....341
5.15.13i2c_transfer_dir_set function....342
5.15.14i2c_transfer_dir_get function....342

5.15.15i2c_matched_addr_get function....343
5.15.16i2c_auto_stop_enable function....343
5.15.17i2c_reload_enable function....343
5.15.18i2c_cnt_set function....344
5.15.19i2c_addr10_header_enable function....344
5.15.20i2c_general_call_enable function....345
5.15.21i2c_smbus_alert_set function....345
5.15.22i2c_slave_data_ctrl_enable function....346
5.15.23i2c_pec_calculate_enable function....346
5.15.24i2c_pec_transmit_enable function....347
5.15.25i2c_pec_value_get function....347
5.15.26i2c_timeout_set function....348
5.15.27i2c_timeout_detcet_set function....348
5.15.28i2c_timeout_enable function....349
5.15.29i2c_ext_timeout_set function....349
5.15.30i2c_ext_timeout_enable function....350
5.15.31i2c_interrupt_enable function....350
5.15.32i2c_interrupt_get function....351
5.15.33i2c_dma_enable function....352
5.15.34i2c_transmit_set function....352
5.15.35i2c_start_generate function....353
5.15.36i2c_stop_generate function....353
5.15.37i2c_data_send function....354
5.15.38i2c_data_receive function....354
5.15.39i2c_flag_get function....354
5.15.40i2c_interrupt_flag_get function....355
5.15.41i2c_flag_clear function....356
5.15.42i2c_config function....357
5.15.43i2c_lowlevel_init function....359
5.15.44i2c_wait_end function....359
5.15.45i2c_wait_flag function....360
5.15.46i2c_master_transmit function....361
5.15.47i2c_master_receive function....362
5.15.48i2c_slave_transmit function....362
5.15.49i2c_slave_receive function....363

5.15.50 i2c_master_transmit_int function 363
5.15.51 i2c_master_receive_int function 364
5.15.52 i2c_slave_transmit_int function 364
5.15.53 i2c_slave_receive_int function 365
5.15.54 i2c_master_transmit_dma function 366
5.15.55 i2c_master_receive_dma function 366
5.15.56 i2c_slave_transmit_dma function 367
5.15.57 i2c_slave_receive_dma function 367
5.15.58 i2c_smbus_master_transmit function 368
5.15.59 i2c_smbus_master_receive function 369
5.15.60 i2c_smbus_slave_transmit function 369
5.15.61 i2c_smbus_slave_receive function 370
5.15.62 i2c_memory_write function 370
5.15.63 i2c_memory_write_int function 371
5.15.64 i2c_memory_write_dma function 372
5.15.65 i2c_memory_read function 373
5.15.66 i2c_memory_read_int function 374
5.15.67 i2c_memory_read_dma function 374
5.15.68 i2c_evt_irq_handler function 375
5.15.69 i2c_err_irq_handler function 376
5.15.70 i2c_dma_tx_irq_handler function 376
5.15.71 i2c_dma_rx_irq_handler function 377
5.16 Nested vectored interrupt controller (NVIC) 378
5.16.1 nvic_system_reset function 378
5.16.2 nvic_irq_enable function 379
5.16.3 nvic_irq_disable function 380
5.16.4 nvic_priority_group_config function 380
5.16.5 nvic_vector_table_set function 381
5.16.6 nvic_lowpower_mode_config function 381
5.17 Power controller (PWC) 382
5.17.1 pwc_reset function 383
5.17.2 pwc_battery_powered_domain_access function 383
5.17.3 pwc_pvm_level_select function 384
5.17.4 pwc_power_voltage_monitor_enable function 384
5.17.5 pwc_wakeup_pin_enable function 385

5.17.6 pwc_flag_clear function....385
5.17.7 pwc_flag_get function....386
5.17.8 pwc_sleep_mode_enter function....386
5.17.9 pwc_deep_sleep_mode_enter function....387
5.17.10 pwc_voltage_regulate_set function....387
5.17.11 pwc_standby_mode_enter function....388
5.17.12 pwc_ldo_output_voltage_set function....388
5.18 Qud-SPI interface (QSPI)....389
5.18.1 qspi_encryption_enable function....390
5.18.2 qspi_sck_mode_set function....390
5.18.3 qspi_clk_division_set function....391
5.18.4 qspi_xip_cache_bypass_set function....392
5.18.5 qspi_interrupt_enable function....392
5.18.6 qspi_interrupt_flag_get function....393
5.18.7 qspi_flag_get function....393
5.18.8 qspi_flag_clear function....394
5.18.9 qspi_dma_rx_threshold_set function....394
5.18.10 qspi_dma_tx_threshold_set function....395
5.18.11 qspi_dma_enable function....395
5.18.12 qspi_busy_config function....396
5.18.13 qspi_xip_enable function....396
5.18.14 qspi_cmd_operation_kick function....397
5.18.15 qspi_xip_init function....399
5.18.16 qspi_byte_read function....401
5.18.17 qspi_half_word_read function....401
5.18.18 qspi_word_read function....402
5.18.19 qspi_word_write function....402
5.18.20 qspi_half_word_write function....403
5.18.21 qspi_byte_write function....403
5.19 System configuration controller (SCFG)....404
5.19.1 scfg_reset function....404
5.19.2 scfg_xmc_mapping_swap_set function....405
5.19.3 scfg_infrared_config function....405
5.19.4 scfg_mem_map_set function....406
5.19.5 scfg_emac_interface_set function....406

5.19.6 scfg_exint_line_config function 407
5.19.7 scfg_pins_ultra_driven_enable function 408
5.20 SDIO interface (SDIO) 409
5.20.1 sdio_reset function 410
5.20.2 sdio_power_set function 411
5.20.3 sdio_power_status_get function 411
5.20.4 sdio_clock_config function 412
5.20.5 sdio_bus_width_config function 412
5.20.6 sdio_clock_bypass function 413
5.20.7 sdio_power_saving_mode_enable function 413
5.20.8 sdio_flow_control_enable function 414
5.20.9 sdio_clock_enable function 414
5.20.10 sdio_dma_enable function 415
5.20.11 sdio_interrupt_enable function 415
5.20.12 sdio_flag_get function 416
5.20.13 sdio_interrupt_flag_get function 417
5.20.14 sdio_flag_clear function 418
5.20.15 sdio_command_config function 419
5.20.16 sdio_command_state_machine_enable function 420
5.20.17 sdio_command_response_get function 420
5.20.18 sdio_response_get function 421
5.20.19 sdio_data_config function 421
5.20.20 sdio_data_state_machine_enable function 423
5.20.21 sdio_data_counter_get function 423
5.20.22 sdio_data_read function 424
5.20.23 sdio_buffer_counter_get function 424
5.20.24 sdio_data_write function 425
5.20.25 sdio_read_wait_mode_set function 425
5.20.26 sdio_read_wait_start function 426
5.20.27 sdio_read_wait_stop function 426
5.20.28 sdio_io_function_enable function 427
5.20.29 sdio_io_suspend_command_set function 427
5.21 Serial peripheral interface (SPI)/I²S 428
5.21.1 spi_i2s_reset function 429
5.21.2 spi_default_para_init function 429

5.21.3 spi_init function....430
5.21.4 spi_ti_mode_enable function....432
5.21.5 spi_crc_next_transmit function....432
5.21.6 spi_crc_polynomial_set function....433
5.21.7 spi_crc_polynomial_get function....433
5.21.8 spi_crc_enable function....434
5.21.9 spi_crc_value_get function....434
5.21.10 spi_hardware_cs_output_enable function....435
5.21.11 spi_software_cs_internal_level_set function....435
5.21.12 spi_frame_bit_num_set function....436
5.21.13 spi_half_duplex_direction_set function....436
5.21.14 spi_enable function....437
5.21.15 i2s_default_para_init function....437
5.21.16 i2s_init function....438
5.21.17 i2s_enable function....439
5.21.18 spi_i2s_interrupt_enable function....440
5.21.19 spi_i2s_dma_transmitter_enable function....441
5.21.20 spi_i2s_dma_receiver_enable function....441
5.21.21 spi_i2s_data_transmit function....442
5.21.22 spi_i2s_data_receive function....442
5.21.23 spi_i2s_flag_get function....443
5.21.24 spi_i2s_interrupt_flag_get function....444
5.21.25 spi_i2s_flag_clear function....445
5.22 SysTick....446
5.22.1 systick_clock_source_config function....446
5.22.2 SysTick_Config function....447
5.23 Timer (TMR)....448
5.23.1 tmr_reset function....450
5.23.2 tmr_counter_enable function....451
5.23.3 tmr_output_default_para_init function....451
5.23.4 tmr_input_default_para_init function....452
5.23.5 tmr_brkdt_default_para_init function....452
5.23.6 tmr_base_init function....453
5.23.7 tmr_clock_source_div_set function....453
5.23.8 tmr_cnt_dir_set function....454

5.23.9 tmr_repetition_counter_set function....455
5.23.10tmr_counter_value_set function....455
5.23.11tmr_counter_value_get function....456
5.23.12tmr_div_value_set function....456
5.23.13tmr_div_value_get function....457
5.23.14tmr_output_channel_config function....457
5.23.15tmr_output_channel_mode_select function....459
5.23.16tmr_period_value_set function....460
5.23.17tmr_period_value_get function....460
5.23.18tmr_channel_value_set function....461
5.23.19tmr_channel_value_get function....461
5.23.20tmr_period_buffer_enable function....462
5.23.21tmr_output_channel_buffer_enable function....462
5.23.22tmr_output_channel_immediately_set function....463
5.23.23tmr_output_channel_switch_set function....464
5.23.24tmr_one_cycle_mode_enable function....464
5.23.25tmr_32_bit_function_enable function....465
5.23.26tmr_overflow_request_source_set function....465
5.23.27tmr_overflow_event_disable function....466
5.23.28tmr_input_channel_init function....466
5.23.29tmr_channel_enable function....468
5.23.30tmr_input_channel_filter_set function....468
5.23.31tmr_pwm_input_config function....469
5.23.32tmr_channel1_input_select function....470
5.23.33tmr_input_channel_divider_set function....470
5.23.34tmr_primary_mode_select function....471
5.23.35tmr_sub_mode_select function....472
5.23.36tmr_channel_dma_select function....472
5.23.37tmr_hall_select function....473
5.23.38tmr_channel_buffer_enable function....473
5.23.39tmr_trgout2_enable function....474
5.23.40tmr_trigger_input_select function....474
5.23.41tmr_sub_sync_mode_set function....475
5.23.42tmr_dma_request_enable function....475
5.23.43tmr_interrupt_enable function....476

5.23.44tmr_interrupt_flag_get function 477
5.23.45tmr_flag_get function 478
5.23.46tmr_flag_clear function 479
5.23.47tmr_event_sw_trigger function 479
5.23.48tmr_output_enable function 480
5.23.49tmr_internal_clock_set function 480
5.23.50tmr_output_channel_polarity_set function 481
5.23.51tmr_external_clock_config function 481
5.23.52tmr_external_clock_mode1_config function 482
5.23.53tmr_external_clock_mode2_config function 483
5.23.54tmr_encoder_mode_config function 483
5.23.55tmr_force_output_set function 484
5.23.56tmr_dma_control_config function 485
5.23.57tmr_brkdt_config function 486
5.23.58tmr_iremap_config function 488
5.24 Universal synchronous/asynchronous receiver/transmitter (USART) 489
5.24.1 usart_reset function 490
5.24.2 usart_init function 491
5.24.3 usart_parity_selection_config function 491
5.24.4 usart_enable function 492
5.24.5 usart_transmitter_enable function 492
5.24.6 usart_receiver_enable function 493
5.24.7 usart_clock_config function 493
5.24.8 usart_clock_enable function 494
5.24.9 usart_interrupt_enable function 494
5.24.10usart_dma_transmitter_enable function 495
5.24.11usart_dma_receiver_enable function 496
5.24.12usart_wakeup_id_set function 496
5.24.13usart_wakeup_mode_set function 496
5.24.14usart_receiver_mute_enable function 497
5.24.15usart_break_bit_num_set function 498
5.24.16usart_lin_mode_enable function 498
5.24.17usart_data_transmit function 499
5.24.18usart_data_receive function 499
5.24.19usart_break_send function 500

5.24.20usart_smartcard_guard_time_set function 500
5.24.21usart_irda_smartcard_division_set function 501
5.24.22usart_smartcard_mode_enable function 501
5.24.23usart_smartcard_nack_set function 502
5.24.24usart_single_line_halfduplex_select function 502
5.24.25usart_irda_mode_enable function....503
5.24.26usart_irda_low_power_enable function 503
5.24.27usart_hardware_flow_control_set function 504
5.24.28usart_flag_get function....504
5.24.29usart_interrupt_flag_get function 505
5.24.30usart_flag_clear function 506
5.24.31usart_rs485_delay_time_config function 506
5.24.32usart_transmit_receive_pin_swap function....507
5.24.33usart_id_bit_num_set function 507
5.24.34usart_de_polarity_set function 508
5.24.35usart_rs485_mode_enable function....508
5.25 Watchdog timer (WDT)....509
5.25.1 wdt_enable function 509
5.25.2 wdt_counter_reload function 510
5.25.3 wdt_reload_value_set function 510
5.25.4 wdt_divider_set function....511
5.25.5 wdt_register_write_enable function 511
5.25.6 wdt_flag_get function 512
5.25.7 wdt_window_counter_set function 512
5.26 Window watchdog timer (WWDT)....513
5.26.1 wwdt_reset function....513
5.26.2 wwdt_divider_set function 514
5.26.3 wwdt_enable function....514
5.26.4 wwdt_interrupt_enable function 515
5.26.5 wwdt_counter_set function....515
5.26.6 wwdt_window_counter_set function 515
5.26.7 wwdt_flag_get function....516
5.26.8 wwdt_interrupt_flag_get function 516
5.26.9 wwdt_flag_clear function....517
5.27 External memory controller (XMC) 518

5.27.1 xmc_nor_sram_reset function 521
5.27.2 xmc_nor_sram_init function 522
5.27.3 xmc_nor_sram_timing_config function 524
5.27.4 xmc_norsram_default_para_init function 527
5.27.5 xmc_norsram_timing_default_para_init function 527
5.27.6 xmc_nor_sram_enable function 528
5.27.7 xmc_ext_timing_config function 529
5.27.8 xmc_nand_reset function 530
5.27.9 xmc_nand_init function 530
5.27.10 xmc_nand_timing_config function 532
5.27.11 xmc_nand_default_para_init function 534
5.27.12 xmc_nand_timing_default_para_init function 534
5.27.13 xmc_nand_enable function 535
5.27.14 xmc_nand_ecc_enable function 536
5.27.15 xmc_ecc_get function 536
5.27.16 xmc_interrupt_enable function 537
5.27.17 xmc_flag_status_get function 537
5.27.18 xmc_interrupt_flag_status_get function 539
5.27.19 xmc_flag_clear function 540
5.27.20 xmc_pccard_reset function 540
5.27.21 xmc_pccard_init function 541
5.27.22 xmc_pccard_timing_config function 543
5.27.23 xmc_pccard_default_para_init function 545
5.27.24 xmc_pccard_timing_default_para_init function 545
5.27.25 xmc_pccard_enable function 546
5.27.26 xmc_sdram_reset function 547
5.27.27 xmc_sdram_init function 547
5.27.28 xmc_sdram_default_para_init function 551
5.27.29 xmc_sdram_cmd function 551
5.27.30 xmc_sdram_status_get function 552
5.27.31 xmc_sdram_refresh_counter_set function 553
5.27.32 xmc_sdram_auto_refresh_set function 553

6 Precautions 554

6.1 Device model replacement 554

6.1.1 KEIL environment....554
6.1.2 IAR environment....555
6.2 Unable to identify IC by JLink software in Keil 557
6.3 How to change HEXT crystal 559

7 Revision history 561

List of tables

Table 1. Summary of macro definitions 66
Table 2. List of abbreviations for peripherals 68
Table 3. Summary of BSP function library files 74
Table 4. Function format description for peripherals 75
Table 5. Summary of ACC registers 76
Table 6. Summary of ACC library functions 76
Table 7. acc_calibration_mode_enable function 77
Table 8. acc_step_set function 77
Table 9. acc_interrupt_enable function 78
Table 10. acc_hicktrim_get function 79
Table 11. acc_hickcal_get function 79
Table 12. acc_write_c1 function 80
Table 13. acc_write_c2 function 80
Table 14. acc_write_c3 function 81
Table 15. acc_read_c1 function 81
Table 16. acc_read_c2 function 82
Table 17. acc_read_c3 function 82
Table 18. acc_flag_get function 83
Table 19. acc_flag_clear function 83
Table 20. acc_flag_clear function 84
Table 21. Summary of ADC registers 85
Tabl e22. Summary of ADC library functions 85
Table 23. adc_reset function 87
Table 24. adc_enable function 87
Table 25. adc_base_default_para_init function 88
Table 26. adc_base_config function 88
Table 27. adc_common_default_para_init function 89
Table 28. adc_common_config function 90
Table 29. adc_resolution_set function 93
Table 30. adc_voltage_battery_enable function 94
Table 31. adc_dma_mode_enable function 94
Table 32. adc_dma_request_repeat_enable function 95
Table 33. adc_interrupt_enable function 95
Table 34. adc_calibration_value_set function 96

Table 35. adc_calibration_init function 96
Table 36. adc_calibration_init_status_get function 97
Table 37. adc_calibration_start function 97
Table 38. adc_calibration_status_get function 98
Table 39. adc_voltage_monitor_enable function 98
Table 40. adc_voltage_monitor_threshold_value_set function 99
Table 41. adc_voltage_monitor_single_channel_select function 100
Table 42. adc_ordinary_channel_set function 100
Table 43. adc_preempt_channel_length_set function 101
Table 44. adc_preempt_channel_set function 102
Table 45. adc_ordinary_conversion_trigger_set function 102
Table 46. adc_preempt_conversion_trigger_set function 104
Table 47. adc_preempt_offset_value_set function 105
Table 48. adc_ordinary_part_count_set function 106
Table 49. adc_ordinary_part_mode_enable function 106
Table 50. adc_preempt _part_mode_enable function 107
Table 51. adc_preempt _auto_mode_enable function 107
Table 52. adc_conversion_stop function 108
Table 53. adc_conversion_stop_status_get function 108
Table 54. adc_occe_each_conversion_enable function 109
Table 55. adc_ordinary_software_trigger_enable function 109
Table 56. adc_ordinary_software_trigger_status_get function 110
Table 57. adc_preempt_software_trigger_enable function 110
Table 58. adc_preempt_software_trigger_status_get function 111
Table 59. adc_ordinary_conversion_data_get function 111
Table 60. adc_combine_ordinary_conversion_data_get function 112
Table 61. adc_preempt_conversion_data_get function 112
Table 62. adc_flag_get function 113
Table 63. adc_interrupt_flag_get function 113
Table 64. adc_flag_clear function 114
Table 65. adc_ordinary_oversample_enable function 114
Table 66. adc_preempt_oversample_enable function 115
Table 67. adc_oversample_ratio_shift_set function 115
Table 68. adc_ordinary_oversample_trig_enable function 116
Table 69. adc_ordinary_oversample_restart_set function 117

Table 70. Summary of CAN registers 118
Table 71. Summary of CAN library functions 119
Table 72. can_reset function 120
Table 73. can_baudrate_default_para_init function 120
Table 74. can_baudrate_set function 121
Table 75. can_default_para_init function 122
Table 76. can_base_init function 122
Table 77. can_filter_default_para_init function 124
Table 78. can_filter_init function 124
Table 79. can_debug_transmission_prohibit function 126
Table 80. can_ttc_mode_enable function 126
Table 81. can_message_transmit function 127
Table 82. can_transmit_status_get function 128
Table 83. can_transmit_cancel function 129
Table 84. can_message_receive function 130
Table 85. can_receive_fifo_release function 131
Table 86. can_receive_message_pending_get function 132
Table 87. can_operating_mode_set function 132
Table 88. can_doze_mode_enter function 133
Table 89. can_doze_mode_exit function 133
Table 90. can_error_type_record_get function 134
Table 91. can_receive_error_counter_get function 135
Table 92. can_transmit_error_counter_get function 135
Table 93. can_interrupt_enable function 136
Table 94. can_flag_get function 137
Table 95. can_interrupt_flag_get function 138
Table 96. can_flag_clear function 139
Table 97. Summary of CRC registers 140
Table 98. Summary of CRC library functions 140
Table 99. crc_data_reset function 141
Table 100. crc_one_word_calculate function 141
Table 101. crc_block_calculate function 142
Table 102. crc_data_get function 142
Table 103. crc_common_data_set function 143
Table 104. crc_common_data_get function 143

Table 105. crc_init_data_set function 144
Table 106. crc_reverse_input_data_set function 144
Table 107. crc_reverse_output_data_set function 145
Table 108. crc_poly_value_set function 145
Table 109. crc_poly_value_get function 146
Table 110. crc_poly_size_set function 146
Table 111. crc_poly_size_get function 147
Table 112. Summary of CRM registers 148
Table 113. Summary of CRM library functions 149
Table 114. crm_reset function 150
Table 115. crm_lext_bypass function 150
Table 116. crm_hext_bypass function 151
Table 117. crm_flag_get function 151
Table 118. crm_interrupt_flag_get function 152
Table 119. crm_hext_stable_wait function 153
Table 120. crm_hick_clock_trimming_set function 153
Table 121. crm_hick_clock_calibration_set function 154
Table 122. crm_periph_clock_enable function 154
Table 123. crm_periph_reset function 155
Table 124. crm_periph_lowpower_mode_enable function 156
Table 125. crm_clock_source_enable function 156
Table 126. crm_flag_clear function 157
Table 127. crm_ertc_clock_select function 158
Table 128. crm_ertc_clock_enable function 158
Table 129. crm_ahb_div_set function 159
Table 130. crm_apb1_div_set function 159
Table 131. crm_apb2_div_set function 160
Table 132. crm_usb_clock_div_set function 160
Table 133. crm_clock_failure_detection_enable function 161
Table 134. crm_battery_powered_domain_reset function 161
Table 135. crm_pll_config function 162
Table 136. crm_sysclk_switch function 163
Table 137. crm_sysclk_switch_status_get function 163
Table 138. crm_clocks_freq_get function 164
Table 139. crm_clock_out1_set function 165

Table 140. crm_clock_out2_set function 165
Table 141. crm_clkout_div_set function 166
Table 142. crm_interrupt_enable function 167
Table 143. crm_auto_step_mode_enable function 167
Table 144. crm_hick_sclk_frequency_select function 168
Table 145. crm_usb_clock_source_select function 168
Table 146. crm_clkout_to_tmr10_enable function 169
Table 147. crm_emac_output_pulse_set function 169
Table 148. crm_pll_parameter_calculate function 170
Table 149. Summary of DAC registers 171
Table 150. Summary of DAC library functions 171
Table 151. dac_reset function 172
Table 152. dac_enable function 172
Table 153. dac_output_buffer_enable function 173
Table 154. dac_trigger_enable function 173
Table 155. dac_trigger_select function 174
Table 156. dac_software_trigger_generate function 174
Table 157. dac_dual_software_trigger_generate function 175
Table 158. dac_wave_generate function 175
Table 159. dac_mask_amplitude_select function 176
Table 160. dac_dma_enable function 176
Table 161. dac_data_output_get function 177
Table 162. dac_1_data_set function 177
Table 163. dac_2_data_set function 178
Table 164. dac_dual_data_set function 179
Table 165. dac_udr_enable function 180
Table 166. dac_udr_flag_get function 180
Table 167. dac_udr_interrupt_flag_get function 181
Table 168. dac_udr_flag_clear function 181
Table 169. Summary of DEBUG registers 182
Table 170. Summary of DEBUG library functions 182
Table 171. debug_device_id_get function 183
Table 172. debug_periph_mode_set function 183
Table 173. Summary of DMA registers 185
Table 174. Summary of DMA library functions 186

Table 175. dma_default_para_init function 187
Table 176. dma_init_struct default values 187
Table 177. dma_init function 188
Table 178. dma_reset function 190
Table 179. dma_data_number_set function 190
Table 180. dma_data_number_get function 191
Table 181. dma_interrupt_enable function 191
Table 182. dma_channel_enable function 192
Table 183. dma_flag_get function 192
Table 184. dma_flag_clear function 194
Table 185. dma_flexible_config function 196
Table 186. DMAMUX channel request source ID 197
Table 187. dmamux_enable function 198
Table 188. dmamux_init function 199
Table 189. dmamux_sync_default_para_init function 199
Table 190. dmamux_sync_init_struct default values 199
Table 191. dmamux_sync_config function 200
Table 192. dmamux_generator_default_para_init function 202
Table 193. dmamux_gen_init_struct default values 202
Table 194. dmamux_generator_config function 202
Table 195. dmamux_sync_interrupt_enable function 204
Table 196. dmamux_generator_interrupt_enable function 205
Table 197. dmamux_sync_flag_get function 205
Table 198. dmamux_sync_flag_clear function 206
Table 199. dmamux_generator_flag_get function 207
Table 200. dmamux_generator_flag_clear function 208
Table 201. Summary of DVP registers 209
Table 202. Summary of CVP library functions 209
Table 203. dvp_reset function 210
Table 204. dvp_capture_enable function 211
Table 205. dvp_capture_mode_set function 211
Table 206. dvp_window_crop_enable function 211
Table 207. dvp_window_crop_set function 212
Table 208. dvp_jpeg_enable function 212
Table 209. dvp_sync_mode_set function 213

Table 210. dvp_sync_code_set function 213
Table 211. dvp_sync_unmask_set function 214
Table 212. dvp_pclk_polarity_set function 214
Table 213. dvp_hsync_polarity_set function 215
Table 214. dvp_vsync_polarity_set function 215
Table 215. dvp_basic_frame_rate_control_set function 216
Table 216. dvp_pixel_data_length_set function 216
Table 217. dvp_enable function 217
Table 218. dvp_zoomout_select function 217
Table 219. dvp_zoomout_set function 218
Table 220. dvp_basic_status_get function 219
Table 221. dvp_interrupt_enable function 219
Table 222. dvp_interrupt_flag_get function 220
Table 223. dvp_flag_get function 220
Table 224. dvp_flag_clear function function 221
Table 225. dvp_enhanced_scaling_resize_enable function 222
Table 226. dvp_enhanced_scaling_resize_set function 222
Table 227. dvp_enhanced_framerate_set function 223
Table 228. dvp_monochrome_image_binarization_set function 223
Table 229. dvp_enhanced_data_format_set function 224
Table 230. dvp_input_data_unused_set function 224
Table 231. dvp_dma_burst_set function 225
Table 232. dvp_sync_event_interrupt_set function 226
Table 233. Summary of EDMA registers 228
Table 234. edma_reset function 230
Table 235. edma_init function function 231
Table 236. edma_default_para_init function 233
Table 237. edma_init_struct default values 233
Table 238. edma_stream_enable function 234
Table 239. edma_interrupt_enable function 235
Table 240. edma_peripheral_inc_offset_set function 235
Table 241. edma_flow_controller_enable function 236
Table 242. edma_data_number_set function 237
Table 243. edma_data_number_get function 237
Table 244. edma_double_buffer_mode_init function 238

Table 245. edma_double_buffer_mode_enable function 238
Table 246. edma_memory_addr_set function 239
Table 247. edma_stream_status_get function 239
Table 248. edma_fifo_status_get function 240
Table 249. edma_flag_get function 240
Table 250. edma_2d_init function 242
Table 251. edma_2d_enable function 243
Table 252. edma_link_list_init function 243
Table 253. edma_link_list_enable function 244
Table 254. edma_flag_clear function 244
Table 255. dmamux_sync_init_struct values 245
Table 256. edmamux_enable function 245
Table 257. edmamux_init function 246
Table 258. EDMAMUX channel request source ID 246
Table 259. edmamux_sync_default_para_init function 248
Table 260. edmamux_sync_init_struct default values 248
Table 261. edmamux_sync_config function 248
Table 262. edmamux_generator_default_para_init function 250
Table 263.edmamux_gen_init_struct default values 250
Table 264. edmamux_generator_config function 251
Table 265. edmamux_sync_interrupt_enable function 252
Table 266. edmamux_generator_interrupt_enable function 253
Table 267. edmamux_sync_flag_get function 253
Table 268. edmamux_sync_flag_clear function 254
Table 269. edmamux_generator_flag_get function 255
Table 270. edmamux_generator_flag_clear function 256
Table 271. Summary of ERTC registers 257
Table 272. Summary of ERTC library functions 257
Table 273. ertc_num_to_bcd function 259
Table 274. ertc_bcd_to_num function 259
Table 275. ertc_write_protect_enable function 260
Table 276. ertc_write_protect_disable function 260
Table 277. ertc_wait_update function 260
Table 278. ertc_wait_flag function 261
Table 279. ertc_init_mode_enter function 261

Table 280. ertc_init_mode_exit function 262
Table 281. ertc_reset function 262
Table 282. ertc_divider_set function 263
Table 283. ertc_hour_mode_set function 263
Table 284. ertc_date_set function 264
Table 285. ertc_time_set function 264
Table 286. ertc_calendar_get function 265
Table 287. ertc_sub_second_get function 266
Table 288. ertc_alarm_mask_set function 266
Table 289. ertc_alarm_week_date_select function 267
Table 290. ertc_alarm_set function 268
Table 291. ertc_alarm_sub_second_set function 269
Table 292. ertc_alarm_enable function 270
Table 293. ertc_alarm_get function 270
Table 294. ertc_alarm_sub_second_get function 272
Table 295. ertc_wakeup_clock_set function 272
Table 296. ertc_wakeup_counter_set function 273
Table 297. ertc_wakeup_counter_get function 273
Table 298. ertc_wakeup_enable function 274
Table 299. ertc_smooth_calibration_config function 274
Table 300. ertc_coarse_calibration_set function 275
Table 301. ertc_coarse_calibration_enable function 275
Table 302. ertc_cal_output_select function 276
Table 303. ertc_cal_output_enable function 276
Table 304. ertc_time_adjust function 277
Table 305. ertc_daylight_set function 278
Table 306. ertc_daylight_bpr_get function 278
Table 307. ertc_refer_clock_detect_enable function 279
Table 308. ertc_direct_read_enable function 279
Table 309. ertc_output_set function 280
Table 310. ertc_timestamp_pin_select function 281
Table 311. ertc_timestamp_valid_edge_set function 281
Table 312. ertc_timestamp_enable function 282
Table 313. ertc_timestamp_get function 282
Table 314. ertc_timestamp_sub_second_get function 283

Table 315. ertc_tamper_1_pin_select function 284
Table 316. ertc_tamper_pull_up_enable function 284
Table 317. ertc_tamper_precharge_set function 285
Table 318. ertc_tamper_filter_set function 286
Table 319. ertc_tamper_detect_freq_set function 286
Table 320. ertc_tamper_valid_edge_set function 287
Table 321. ertc_tamper_timestamp_enable function 288
Table 322. ertc_tamper_enable function 288
Table 323. ertc_interrupt_enable function 289
Table 324. ertc_interrupt_get function 289
Table 325. ertc_flag_get function 290
Table 326. ertc_interrupt_flag_get function 291
Table 327. ertc_flag_clear function 291
Table 328. ertc_bpr_data_write function 292
Table 329. ertc_bpr_data_read function 293
Table 330. Summary of EXINT registers 294
Table 331. Summary of EXINT library functions 294
Table 332. exint_reset function 294
Table 333. exint_default_para_init function 295
Table 334. exint_init function 295
Table 335. exint_flag_clear function 296
Table 336. exint_flag_get function 297
Table 337. exint_interrupt_flag_get function 297
Table 338. exint_software_interrupt_event_generate function 298
Table 339. exint_interrupt_enable function 298
Table 340. exint_event_enable function 299
Table 341. Summary of FLASH registers 299
Table 342. Summary of FLASH library functions 300
Table 343. flash_flag_get function 301
Table 344. flash_flag_clear function 302
Table 345. flash_operation_status_get function 302
Table 346. flash_bank1_operation_status_get function 303
Table 347. flash_bank2_operation_status_get function 303
Table 348. flash_operation_wait_for function 304
Table 349. flash_bank1_operation_wait_for function 304

Table 350. flash_bank2_operation_wait_for function 305
Table 351. flash_unlock function 305
Table 352. flash_bank1_unlock function 306
Table 353. flash_bank2_unlock function 306
Table 354. flash_lock function 306
Table 355. flash_bank1_lock function 307
Table 356. flash_bank2_lock function 307
Table 357. flash_sector_erase function 307
Table 358. flash_block_erase function 308
Table 359. flash_internal_all_erase function 308
Table 360. flash_bank1_erase function 309
Table 361. flash_bank2_erase function 309
Table 362. flash_user_system_data_erase function 310
Table 363. flash_eopb0_config function 310
Table 364. flash_word_program function 311
Table 365. flash_halfword_program function 311
Table 366. flash_byte_program function 312
Table 367. flash_user_system_data_program function 313
Table 368. flash_epp_set function 313
Table 369. flash_epp_status_get function 314
Table 370. flash_fap_enable function 315
Table 371. flash_fap_status_get function 315
Table 372. flash_ssb_set function 316
Table 373. flash_ssb_status_get function 317
Table 374. flash_interrupt_enable function 317
Table 375. flash_slib_enable function 318
Table 376. flash_slib_disable function 318
Table 377. flash_slib_remaining_count_get function 319
Table 378. flash_slib_state_get function 319
Table 379. flash_slib_start_sector_get function 320
Table 380. flash_slib_inststart_sector_get function 320
Table 381. flash_slib_end_sector_get function 321
Table 382. flash_crc_calibrate function 321
Table 383. flash_nzw_boost_enable function 322
Table 384. flash_continue_read_enable function 322

Table 385. Summary of GPIO registers 323
Table 386. Summary of GPIO and IOMUX library functions 323
Table 387. gpio_reset function 324
Table 388. gpio_init function 324
Table 389. gpio_default_para_init function 326
Table 390. gpio_init_struct default values 326
Table 391. gpio_input_data_bit_read function 326
Table 392. gpio_input_data_read function 327
Table 393. gpio_output_data_bit_read function 327
Table 394. gpio_output_data_read function 328
Table 395. gpio_bits_set function 328
Table 396. gpio_bits_reset function 329
Table 397. gpio_bits_write function 329
Table 398. gpio_bits_toggle function 330
Table 399. gpio_port_write function 330
Table 400. gpio_pin_wp_config function 331
Table 401. gpio_pins_huge_driven_config function 331
Table 402. gpio_pin_mux_config function 332
Table 403. Summary of I2C registers 333
Table 404. Summary of I2C library functions 333
Table 405. I2C application-layer library functions 334
Table 406. i2c_reset function 335
Table 407. i2c_init function 336
Table 408. i2c_own_address1_set function 336
Table 409. i2c_own_address2_set function 337
Table 410. i2c_own_address2_enable function 337
Table 411. i2c_smbus_enable function 338
Table 412. i2c_enable function 339
Table 413. i2c_clock_stretch_enable function 339
Table 414. i2c_ack_enable function 340
Table 415. i2c_addr10_mode_enable function 340
Table 416. i2c_transfer_addr_set function 341
Table 417. i2c_transfer_addr_get function 341
Table 418. i2c_transfer_dir_set function 342
Table 419. i2c_transfer_dir_get function 342

Table 420. i2c_matched_addr_get function 343
Table 421. i2c_auto_stop_enable function 343
Table 422. i2c_reload_enable function 343
Table 423. i2c_cnt_set function 344
Table 424. i2c_addr10_header_enable function 344
Table 425. i2c_general_call_enable function 345
Table 426. i2c_smbus_alert_set function 345
Table 427. i2c_slave_data_ctrl_enable function 346
Table 428. i2c_pec_calculate_enable function 346
Table 429. i2c_pec_transmit_enable function 347
Table 430. i2c_pec_value_get function 347
Table 431. i2c_timeout_set function 348
Table 432. i2c_timeout_detcet_set function 348
Table 433. i2c_timeout_enable function 349
Table 434. i2c_ext_timeout_set function 349
Table 435. i2c_ext_timeout_enable function 350
Table 436. i2c_interrupt_enable function 350
Table 437. i2c_interrupt_get function 351
Table 438. i2c_dma_enable function 352
Table 439. i2c_transmit_set function 352
Table 440. i2c_start_generate function 353
Table 441. i2c_stop_generate function 353
Table 442. i2c_data_send function 354
Table 443. i2c_data_receive function 354
Table 444. i2c_flag_get function 354
Table 445. i2c_interrupt_flag_get function 355
Table 446. i2c_flag_clear function 356
Table 447. i2c_config function 357
Table 448. i2c_lowlevel_init function 359
Table 449. i2c_wait_end function 359
Table 450. i2c_wait_flag function 360
Table 451. i2c_master_transmit function 361
Table 452. i2c_master_receive function 362
Table 453. i2c_slave_transmit function 362
Table 454. i2c_slave_receive function 363

Table 455. i2c_master_transmit_int function 363
Table 456. i2c_master_receive_int function 364
Table 457. i2c_slave_transmit_int function 364
Table 458. i2c_slave_receive_int function 365
Table 459. i2c_master_transmit_dma function 366
Table 460. i2c_master_receive_dma function 366
Table 461. i2c_slave_transmit_dma function 367
Table 462. i2c_slave_receive_dma function 367
Table 463. i2c_smbus_master_transmit function 368
Table 464. i2c_smbus_master_receive function 369
Table 465. i2c_smbus_slave_transmit function 369
Table 466. i2c_smbus_slave_receive function 370
Table 466. i2c_memory_write function 370
Table 467. i2c_memory_write_int function 371
Table 468. i2c_memory_write_dma function 372
Table 469. i2c_memory_read function 373
Table 470. i2c_memory_read_int function 374
Table 471. i2c_memory_read_dma function 374
Table 473. i2c_evt_irq_handler function 375
Table 474. i2c_err_irq_handler function 376
Table 475. i2c_dma_tx_irq_handler function 376
Table 476. i2c_dma_rx_irq_handler function 377
Table 477. Summary of NVIC registers 378
Table 478. Summary of NVIC library functions 378
Table 479. nvic_system_reset function 378
Table 480. nvic_irq_enable function 379
Table 481. nvic_irq_disable function 380
Table 482. nvic_priority_group_config function 380
Table 483. nvic_vector_table_set function 381
Table 484. nvic_lowpower_mode_config function 381
Table 485. Summary of PWC registers 382
Table 486. Summary of PWC library functions 382
Table 487. pwc_reset function 383
Table 488. pwc_battery_powered_domain_access function 383
Table 489. pwc_pvm_level_select function 384

Table 490. pwc_power_voltage_monitor_enable function 384
Table 491. pwc_wakeup_pin_enable function 385
Table 492. pwc_flag_clear function 385
Table 493. pwc_flag_get function 386
Table 494. pwc_sleep_mode_enter function 386
Table 495. pwc_deep_sleep_mode_enter function 387
Table 496. pwc_voltage_regulate_set function 387
Table 497. pwc_standby_mode_enter function 388
Table 498. pwc_ldo_output_voltage_set function 388
Table 499. Summary of QSPI registers 389
Table 500. Summary of QSPI library functions 389
Table 501. qspi_encryption_enable function function 390
Table 502. qspi_sck_mode_set function 390
Table 503. qspi_clk_division_set function 391
Table 504. qspi_xip_cache_bypass_set function 392
Table 505. qspi_interrupt_enable function 392
Table 506. qspi_flag_get function 393
Table 507. qspi_flag_get function 393
Table 508. qspi_flag_clear function 394
Table 509. qspi_dma_rx_threshold_set function 394
Table 510. qspi_dma_tx_threshold_set function 395
Table 511. qspi_dma_enable function 395
Table 512. qspi_busy_config function 396
Table 513. qspi_xip_enable function 396
Table 514. qspi_cmd_operation_kick function 397
Table 515. qspi_xip_init function 399
Table 516. qspi_byte_read function 401
Table 517. qspi_half_word_read function 401
Table 518. qspi_word_read function 402
Table 519. qspi_word_write function 402
Table 520. qspi_half_word_write function 403
Table 521. qspi_byte_write function 403
Table 522. Summary of SCFG registers 404
Table 523. Summary of SCFG library functions 404
Table 524. scfg_reset function 404

Table 525. scfg_xmc_mapping_swap_set function 405
Table 526. scfg_infrared_config function 405
Table 527. scfg_mem_map_set function 406
Table 528. scfg_emac_interface_set function 406
Table 529. scfg_exint_line_config function 407
Table 530. scfg_pins_ultra_driven_enable function 408
Table 531. Summary of SDIO registers 409
Table 532. Summary of SDIO library functions 409
Table 533. sdio_reset function 410
Table 534. sdio_power_set function 411
Table 535. sdio_power_status_get function 411
Table 536. sdio_clock_config function 412
Table 537. sdio_bus_width_config function 412
Table 538. sdio_clock_bypass function 413
Table 539. sdio_power_saving_mode_enable function 413
Table 540. sdio_flow_control_enable function 414
Table 541. sdio_clock_enable function 414
Table 542. sdio_dma_enable function 415
Table 543. crm_flag_clear function 415
Table 544. sdio_flag_get function 416
Table 545. sdio_interrupt_flag_get function 417
Table 546. sdio_flag_clear function 418
Table 547. sdio_command_config function 419
Table 548. sdio_command_state_machine_enable function 420
Table 549. sdio_command_response_get function 420
Table 550. sdio_response_get function 421
Table 551. sdio_data_config function 421
Table 552. sdio_data_state_machine_enable function 423
Table 553. sdio_data_counter_get function 423
Table 554. sdio_data_read function 424
Table 555. sdio_buffer_counter_get function 424
Table 556. sdio_data_write function 425
Table 557. sdio_read_wait_mode_set function 425
Table 558. sdio_read_wait_start function 426
Table 559. sdio_read_wait_stop function 426

Table 560. sdio_io_function_enable function 427
Table 561. sdio_io_suspend_command_set function 427
Table 562. Summary of SPI registers 428
Table 563. Summary of SPI library functions 428
Table 564. spi_i2s_reset function 429
Table 565. spi_default_para_init function 429
Table 566. spi_init function 430
Table 567. spi_ti_mode_enable function 432
Table 568. spi_crc_next_transmit function 432
Table 569. spi_crc_polynomial_set function 433
Table 570. spi_crc_polynomial_get function 433
Table 571. spi_crc_enable function 434
Table 572. spi_crc_value_get function 434
Table 573. spi_hardware_cs_output_enable function 435
Table 574. spi_software_cs_internal_level_set function 435
Table 575. spi_frame_bit_num_set function 436
Table 576. spi_half_duplex_direction_set function 436
Table 577. spi_enable function 437
Table 578. i2s_default_para_init function 437
Table 579. i2s_init function 438
Table 580. i2s_enable function 439
Table 581. spi_i2s_interrupt_enable function 440
Table 582. spi_i2s_dma_transmitter_enable function 441
Table 583. spi_i2s_dma_receiver_enable function 441
Table 584. spi_i2s_data_transmit function 442
Table 585. spi_i2s_data_receive function 442
Table 586. spi_i2s_flag_get function 443
Table 587. spi_i2s_flag_clear function 444
Table 588. spi_i2s_flag_clear function 445
Table 589. Summary of SysTick registers 446
Table 590. Summary of SysTick library functions 446
Table 591. systick_clock_source_config function 446
Table 592. SysTick_Config function 447
Table 593. Summary of TMR registers 448
Table 594. Summary of TMR library functions 449

Table 595. tmr_reset function 450
Table 596. tmr_counter_enable function 451
Table 597. tmr_output_default_para_init function 451
Table 598. tmr_output_struct default values 451
Table 599. tmr_input_default_para_init function 452
Table 600. tmr_input_struct default values 452
Table 601. tmr_brkdt_default_para_init function 452
Table 602. tmr_brkdt_struct default values 452
Table 603. tmr_base_init function 453
Table 604. tmr_clock_source_div_set function 453
Table 605. tmr_cnt_dir_set function 454
Table 606. tmr_repetition_counter_set function 455
Table 607. tmr_counter_value_set function 455
Table 608. tmr_counter_value_get function 456
Table 609. tmr_div_value_set function 456
Table 610. tmr_div_value_get function 457
Table 611. tmr_output_channel_config function 457
Table 612. tmr_output_channel_mode_select function 459
Table 613. tmr_period_value_set function 460
Table 614. tmr_period_value_get function 460
Table 615. tmr_channel_value_set function 461
Table 616. tmr_channel_value_get function 461
Table 617. tmr_period_buffer_enable function 462
Table 618. tmr_output_channel_buffer_enable function 462
Table 619. tmr_output_channel_immediately_set function 463
Table 620. tmr_output_channel_switch_set function 464
Table 621. tmr_one_cycle_mode_enable function 464
Table 622. tmr_32_bit_function_enable function 465
Table 623. tmr_overflow_request_source_set function 465
Table 624. tmr_overflow_event_disable function 466
Table 625. tmr_input_channel_init function 466
Table 626. tmr_channel_enable function 468
Table 627. tmr_input_channel_filter_set function 468
Table 628. tmr_pwm_input_config function 469
Table 629. tmr_channel1_input_select function 470

Table 630. tmr_input_channel_divider_set function 470
Table 631. tmr_primary_mode_select function 471
Table 632. tmr_sub_mode_select function 472
Table 633. tmr_channel_dma_select function 472
Table 634. tmr_hall_select function 473
Table 635. tmr_channel_buffer_enable function 473
Table 636. tmr_trgout2_enable function 474
Table 637. tmr_trigger_input_select function 474
Table 638. tmr_sub_sync_mode_set function 475
Table 639. tmr_dma_request_enable function 475
Table 640. tmr_interrupt_enable function 476
Table 641. tmr_interrupt_flag_get function 477
Table 642. tmr_flag_get function 478
Table 643. tmr_flag_clear function 479
Table 644. tmr_event_sw_trigger function 479
Table 645. tmr_output_enable function 480
Table 646. tmr_internal_clock_set function 480
Table 647. tmr_output_channel_polarity_set function 481
Table 648. tmr_external_clock_config function 481
Table 649. tmr_external_clock_mode1_config function 482
Table 650. tmr_external_clock_mode2_config function 483
Table 651. tmr_encoder_mode_config function 483
Table 652. tmr_force_output_set function 484
Table 653. tmr_dma_control_config function 485
Table 654. tmr_brkdt_config function 486
Table 655. tmr_iremap_config function 488
Table 656. Summary of USART registers 489
Table 657. Summary of USART library functions 489
Table 658. usart_reset function 490
Table 659. usart_init function 491
Table 660. usart_parity_selection_config function 491
Table 661. usart_enable function 492
Table 662. usart_transmitter_enable function 492
Table 663. usart_receiver_enable function 493
Table 664. usart_clock_config function 493

Table 665. usart_clock_enable function 494
Table 666. usart_interrupt_enable function 494
Table 667. usart_dma_transmitter_enable function 495
Table 668. usart_dma_receiver_enable function 496
Table 669. usart_wakeup_id_set function 496
Table 670. usart_wakeup_mode_set function 497
Table 671. usart_receiver_mute_enable function 497
Table 672. usart_break_bit_num_set function 498
Table 673. usart_lin_mode_enable function 498
Table 674. usart_data_transmit function 499
Table 675. usart_data_receive function 499
Table 676. usart_break_send function 500
Table 677. usart_smartcard_guard_time_set function 500
Table 678. usart_irda_smartcard_division_set function 501
Table 679. usart_smartcard_mode_enable function 501
Table 680. usart_smartcard_nack_set function 502
Table 681. usart_single_line_halfduplex_select function 502
Table 682. usart_irda_mode_enable function 503
Table 683. usart_irda_low_power_enable function 503
Table 684. usart_hardware_flow_control_set function 504
Table 685. usart_flag_get function 504
Table 686. usart_interrupt_flag_get function 505
Table 687. usart_flag_clear function 506
Table 688. usart_rs485_delay_time_config function 506
Table 689. usart_transmit_receive_pin_swap function 507
Table 690. usart_id_bit_num_set function 507
Table 691. usart_de_polarity_set function 508
Table 692. usart_rs485_mode_enable function 508
Table 693. Summary of WDT registers 509
Table 694. Summary WDT library functions 509
Table 695. wdt_enable function 509
Table 696. wdt_counter_reload function 510
Table 697. wdt_reload_value_set function 510
Table 698. wdt_divider_set function 511
Table 699. wdt_register_write_enable function 511

Table 700. wdt_flag_get function 512
Table 701. wdt_window_counter_set function 512
Table 702. Summary of WWDT registers 513
Table 703. Summary of WWDT library functions 513
Table 704. wwdt_reset function 513
Table 705. wwdt_divider_set function 514
Table 706. wwdt_enable function 514
Table 707. wwdt_interrupt_enable function 515
Table 708. wwdt_counter_set function 515
Table 709. wwdt_window_counter_set function 515
Table 710. wwdt_flag_get function 516
Table 711. wwdt_ interrupt_flag_get function 516
Table 712. wwdt_flag_clear function 517
Table 713. Summary of XMC registers 519
Table 714. Summary of XMC library functions 520
Table 715. xmc_nor_sram_reset function 521
Table 716. xmc_nor_sram_init function 522
Table 717.xmc_nor_sram_timing_config function 524
Table 718. xmc_norsram_default_para_init function 527
Table 719.xmc_nor_sram_init_struct default values 527
Table 720.xmc_norsram_timing_default_para_init function 527
Table 721. xmc_rw_timing_struct and xmc_w_timing_struct default values 528
Table 722. xmc_nor_sram_enable function 528
Table 723. xmc_ext_timing_config function 529
Table 724. xmc_nand_reset function 530
Table 725. xmc_nand_init function 530
Table 726. xmc_nand_timing_config function 532
Table 727. xmc_nand_default_para_init function 534
Table 728. xmc_nand_init_struct default values 534
Table 729. xmc_nand_timing_default_para_init function 534
Table 730. xmc_common_spacetiming_struct and xmc_attribute_spacetiming_struct default values 535
Table 731. xmc_nand_enable function 535
Table 732. xmc_nand_ecc_enable function 536
Table 733. xmc_ecc_get function 536

Table 734. xmc_interrupt_enable function 537
Table 735. xmc_flag_status_get function 537
Table 736. xmc_flag_clear function 539
Table 737. xmc_flag_clear function 540
Table 738. xmc_pccard_reset function 540
Table 739. xmc_pccard_init function 541
Table 740. xmc_nand_pccard_config function 543
Table 741. xmc_pccard_default_para_init function 545
Table 742.xmc_pccard_init_struct default values 545
Table 743. xmc_pccard_timing_default_para_init function 545
Table 744.xmc_common_spacetiming_struct and xmc_attribute_spacetiming_struct default values 546
Table 745. xmc_pccard_enable function 546
Table 746. xmc_sdram_reset function 547
Table 747. xmc_sdram_init function 547
Table 748. xmc_sdram_ default_para_init function 551
Table 749. xmc_sdram_cmd function 551
Table 750. xmc_sdram_status_get function 552
Table 751. xmc_sdram_refresh_counter_set function 553
Table 752. xmc_sdram_auto_refresh_set function 553
Table 753. Clock configuration guideline 560
Table 754. Document revision history 561

List of figures

Figure 1. Pack kit 49
Figure 2. IAR Pack installation window 49
Figure 3. IAR Pack installation 50
Figure 4. View IAR Pack installation status 51
Figure 5. View Keil_v5 Pack installation status 52
Figure 6. Keil_v4 Pack installation window 53
Figure 7. Keil_v4 Pack installation process 53
Figure 8. Keil_v4 Pack installation complete 54
Figure 9. View Keil_v4 Pack installation status 55
Figure 10. Segger pack installation window 56
Figure 11. Segger pack installation process 56
Figure 12. Open J-Flash 57
Figure 13. Create a new project using J-Flash 57
Figure 14. View Device information 58
Figure 15. Keil algorithm file settings 59
Figure 16. Keil algorithm file configuration 60
Figure 17. Select algorithm files using Keil 60
Figure 18. Add algorithm files using Keil 61
Figure 19. IAR project name 62
Figure 20. IAR algorithm file configuration 62
Figure 21. IAR Flash Loader Overview 63
Figure 22. IAR Flash Loader configuration 63
Figure 23. IAR Flash Loader configuration success 64
Figure 24. Template content 65
Figure 25. Keil_v5 template project example 66
Figure 26. Peripheral enable macro definitions 67
Figure 27. BSP folder structure 73
Figure 28. BSP function library structure 74
Figure 29. Change device part number in Keil 554
Figure 30. Change macro definition in Keil 555
Figure 31. Change device part number in IAR 556
Figure 32. Change macro definition in IAR 557
Figure 33. Error warning 1 557
Figure 34. Error warning 2 558

Figure 35. Error warning 3 558
Figure 36. JLinkLog and JLinkSettings 558
Figure 37. Unspecified Cortex–M4 559
Figure 38. AT32_New_Clock_Configuration window 560

1 Overview

In order to allow users to use Artery MCU in an efficiently and quickly manner, we provide a complete set of BSP & Pack tools for the sake of application 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.

This application note gives a detailed account of how to use BSP and Pack.

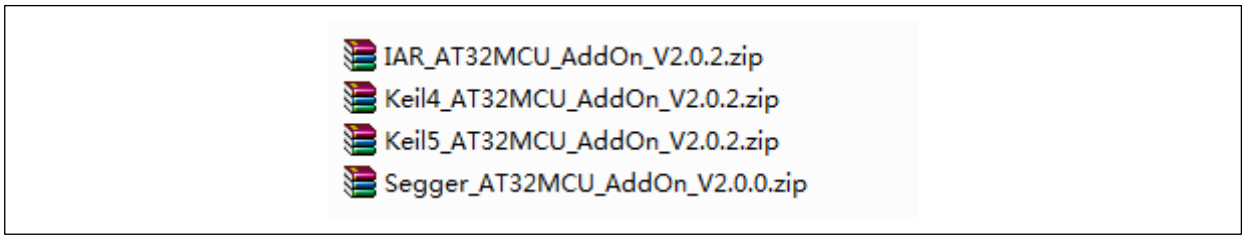
2 How to install Pack

ArteryTek provides a complete set of Pack documents that support various development environments such as Keil_v5, Keil_v4, IAR_v6, IAR_v7 and IAR_v8. Double-click on the corresponding Pack to finish one-stop installation.

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

The pack installation documents are as follows (the specific version information is subject to the actual conditions):

Figure 1. Pack kit



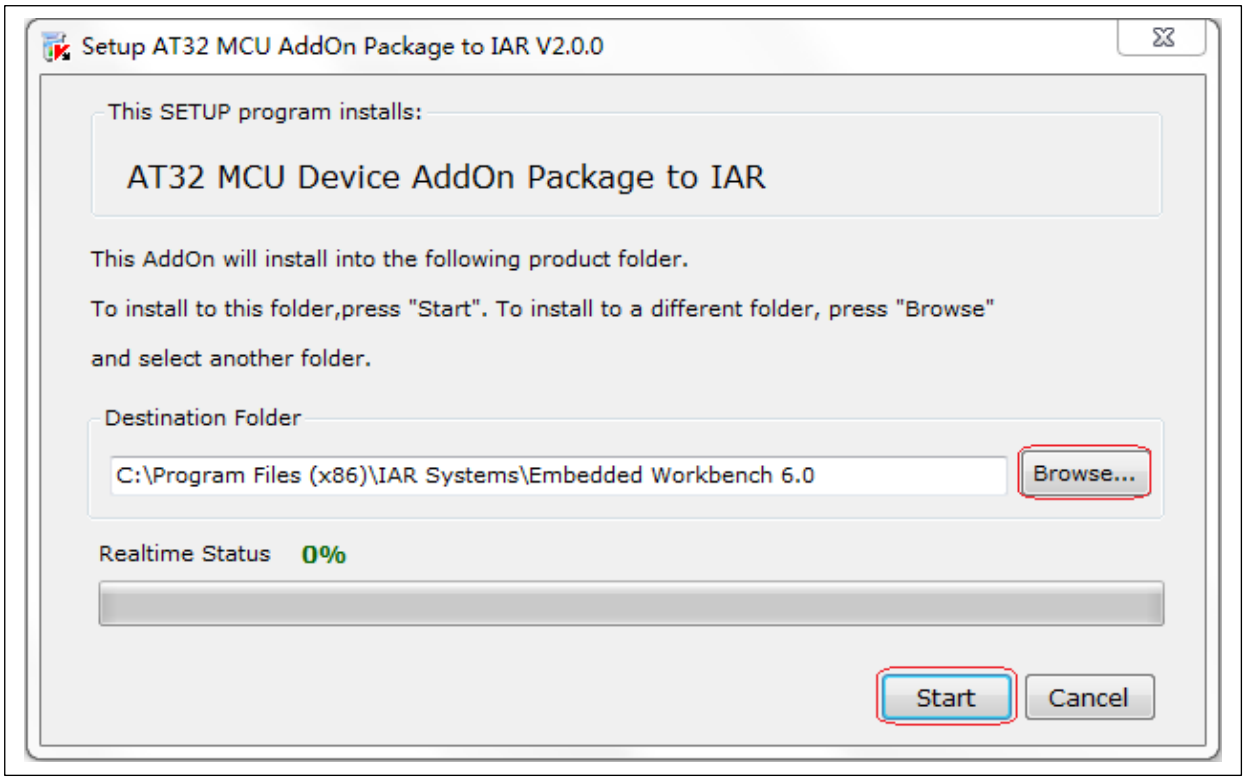
2.1 IAR Pack installation

I**AR_AT32MCU_AddOn.zip**: This is a zip file supporting IAR_V6, IAR_V7 and IAR_V8. Follow the steps below to install:

① Unzip *IAR_AT32MCU_AddOn.zip*

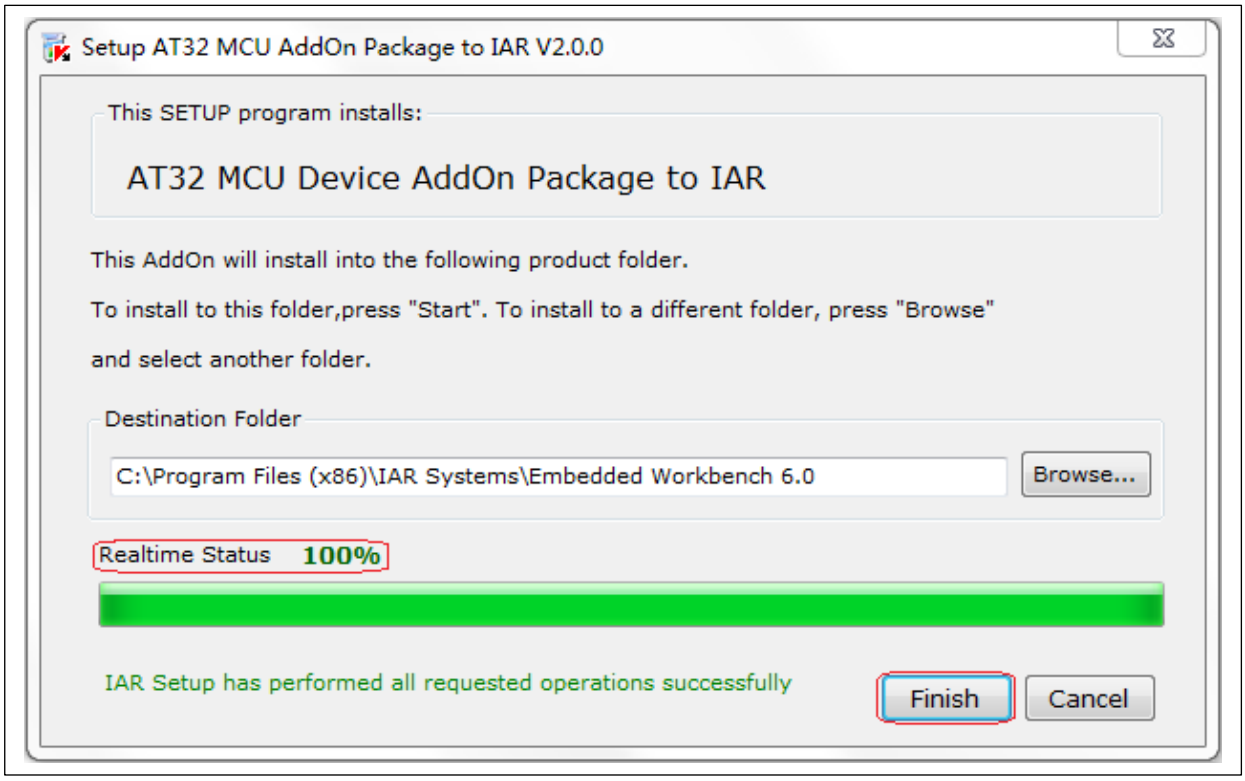
② Double click on *IAR_AT32MCU_AddOn.exe*, and a dialogue box pops up (the specific version information is subject to the actual conditions).

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

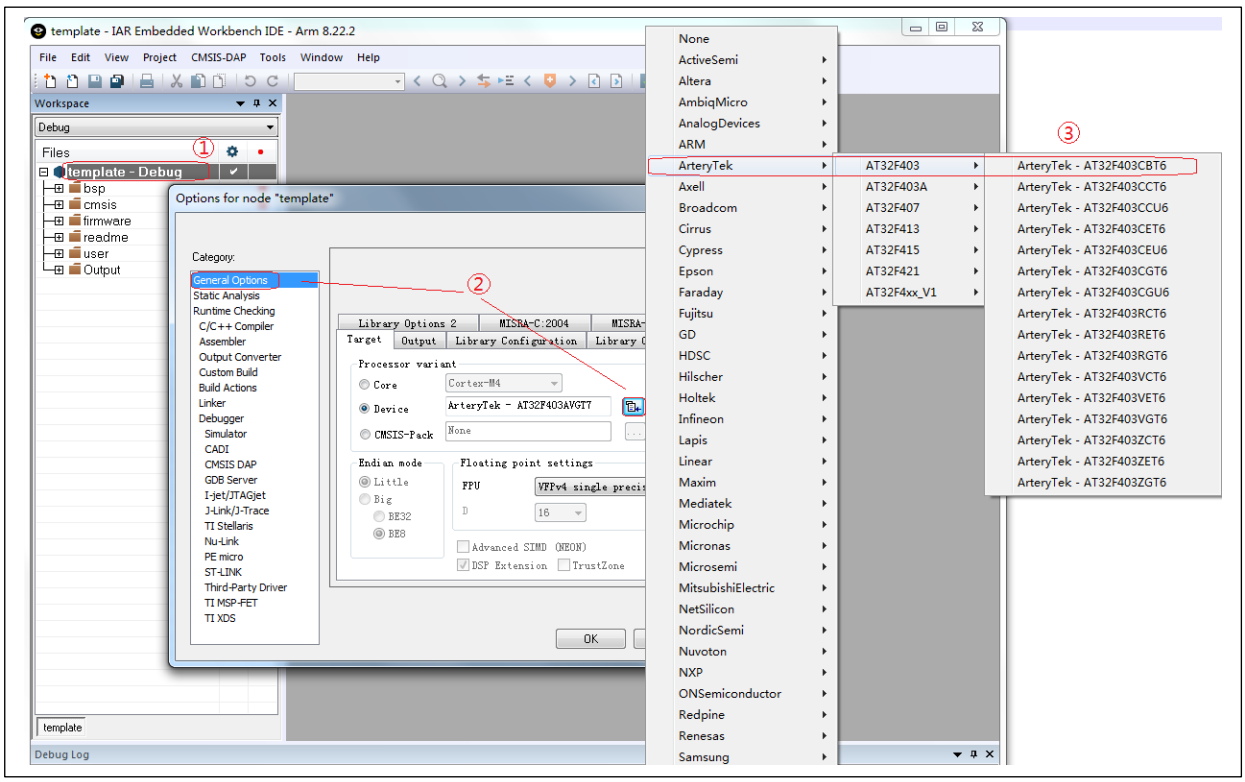


③ Click on “Finish”

④ To check whether the IAR Pack is installed successfully or not, open an IAR project and follow the steps below:

- Right click on a project name, and select “Options…”
- Select “General Options”, and click on the check box
- Click on “ArteryTek” and view AT32 MCU-related information.

Figure 4. View IAR Pack installation status



2.2 Keil_v5 Pack installation

Keil5_AT32MCU_AddOn.zip: This is a zip file supporting Keil_v5. Follow the steps below to install:

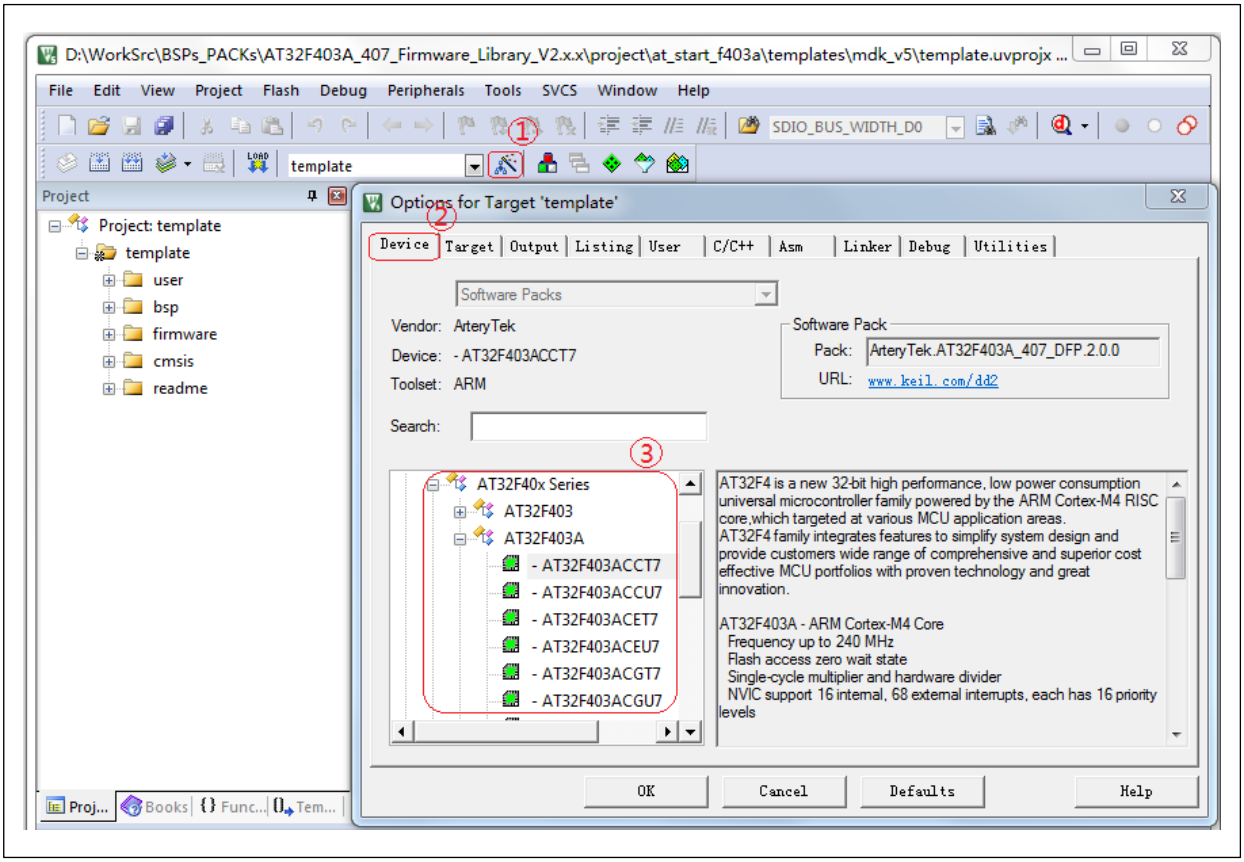
① Unzip *Keil5_AT32MCU_AddOn.zip*. This zip file includes all Keil5 packs supported, all of which are standard Keil_v5 DFP installation files.

② Select the desired Pack, and double click on *ArteryTek.AT32xxxx_DFP.2.x.x.pack* to get one-stop installation.

⑤ To check whether the Keil_v5 Pack is installed successfully or not, follow the steps below:

- Click on wand
- Select “Device”
- View AT32 MCU-related information

Figure 5. View Keil_v5 Pack installation status



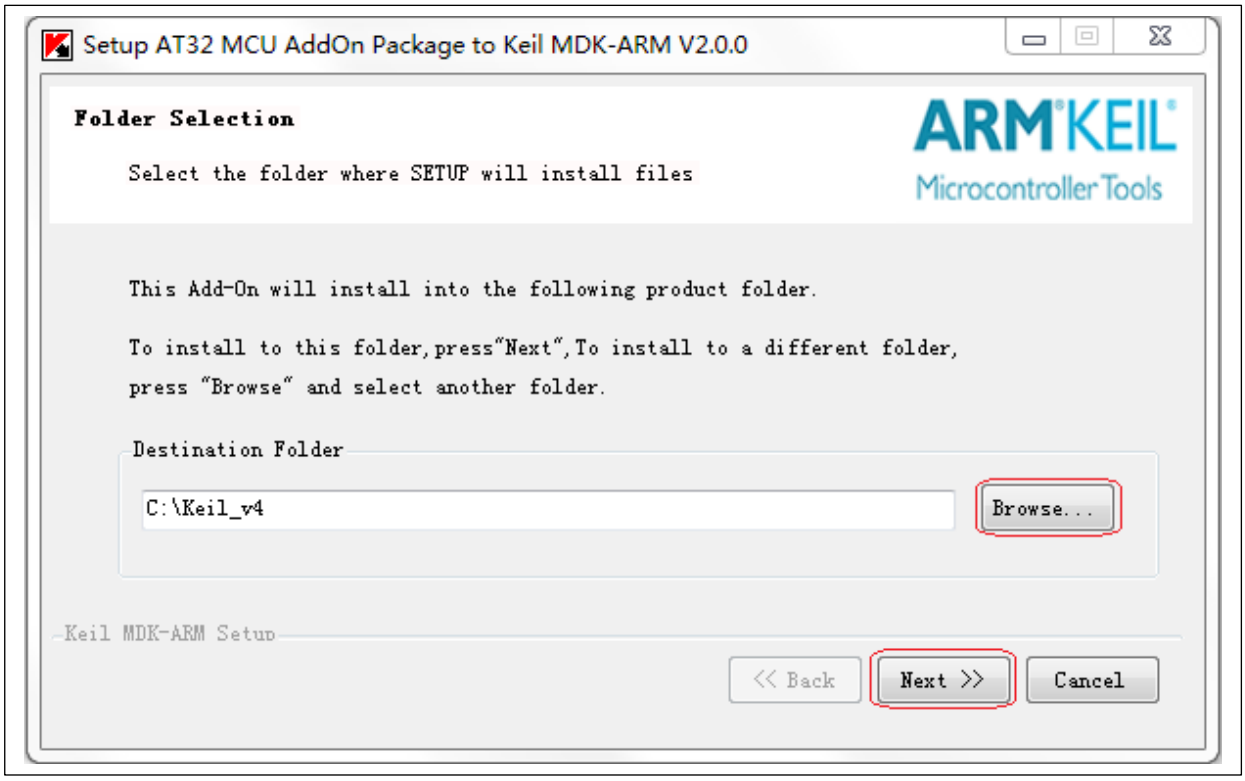
2.3 Keil_v4 Pack installation

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

① Unzip *Keil4_AT32MCU_AddOn.zip*

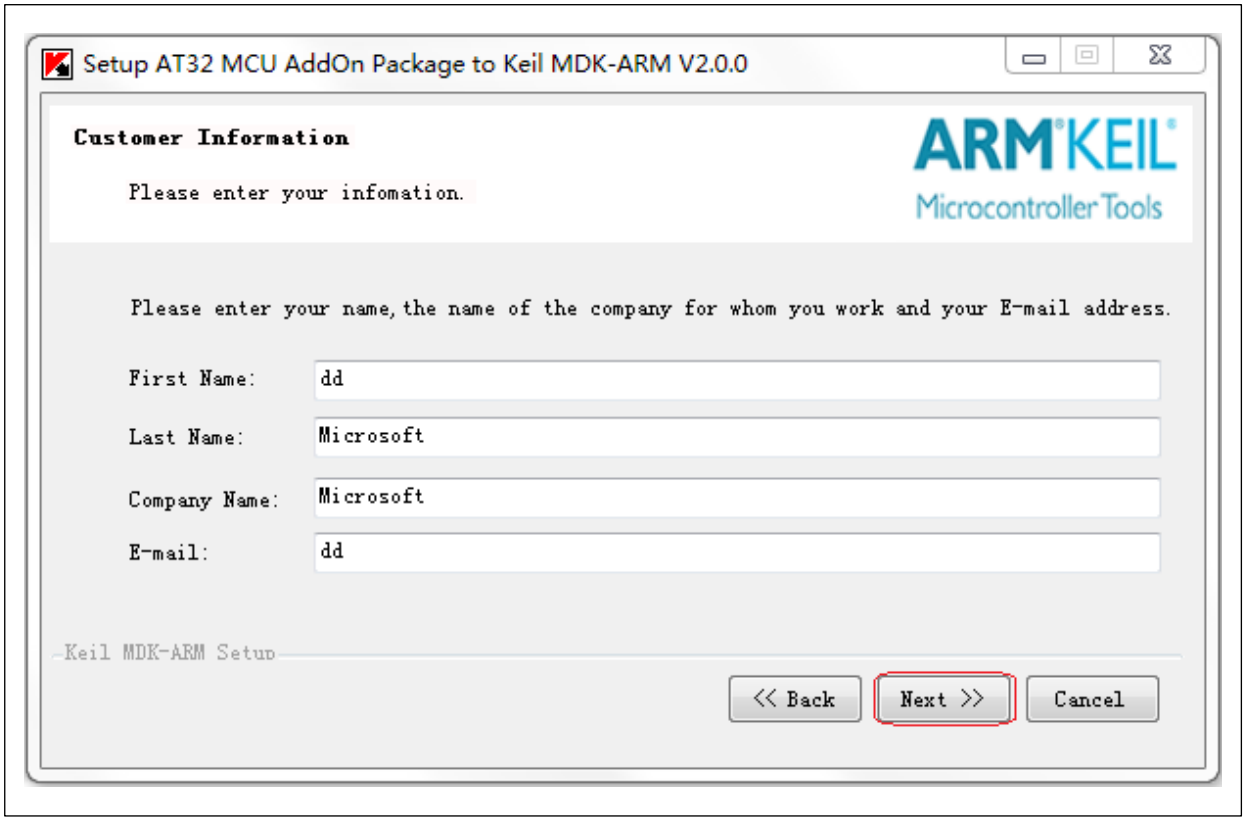
② Double click on *Keil4_AT32MCU_AddOn.exe*, and a dialog box pops up below (the specific version information is subject to the actual conditions).

Figure 6. Keil_v4 Pack installation window



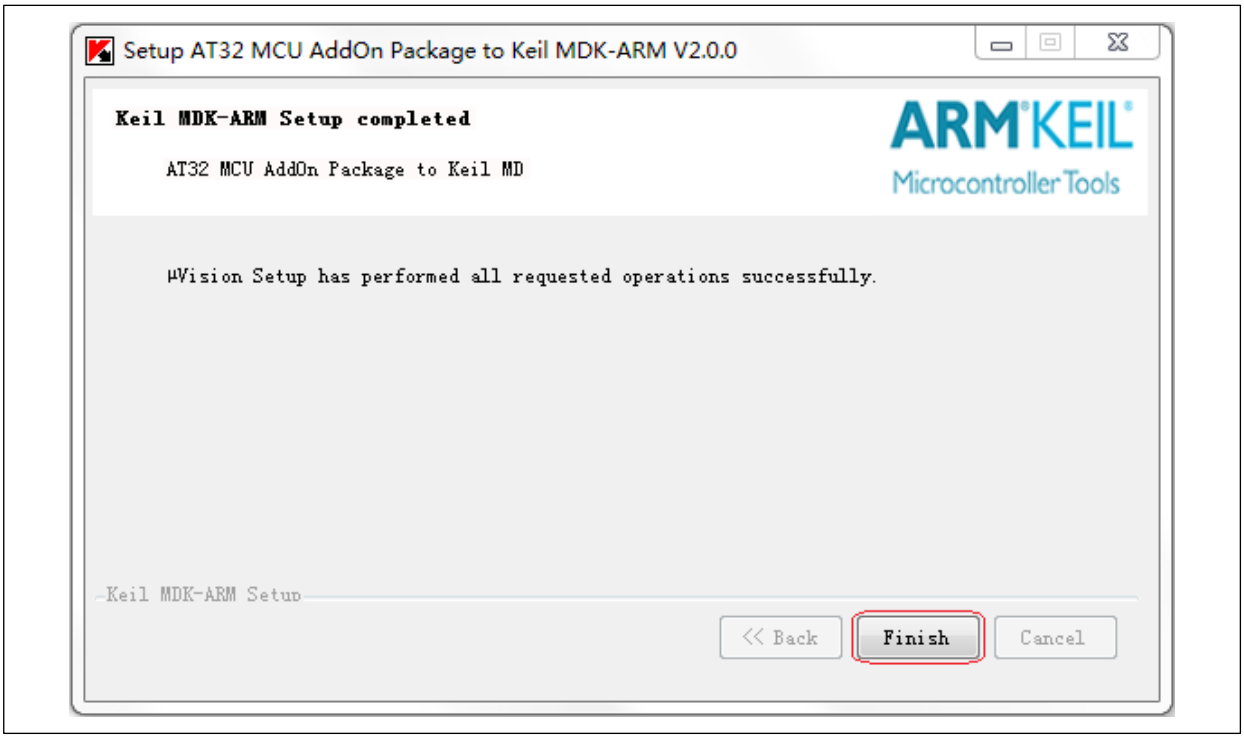
③ If the installation path of Keil_v4 does not match the “Destination Folder”, click on “Browse” to select the actual correct path, then click on “Next”, as shown below.

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.

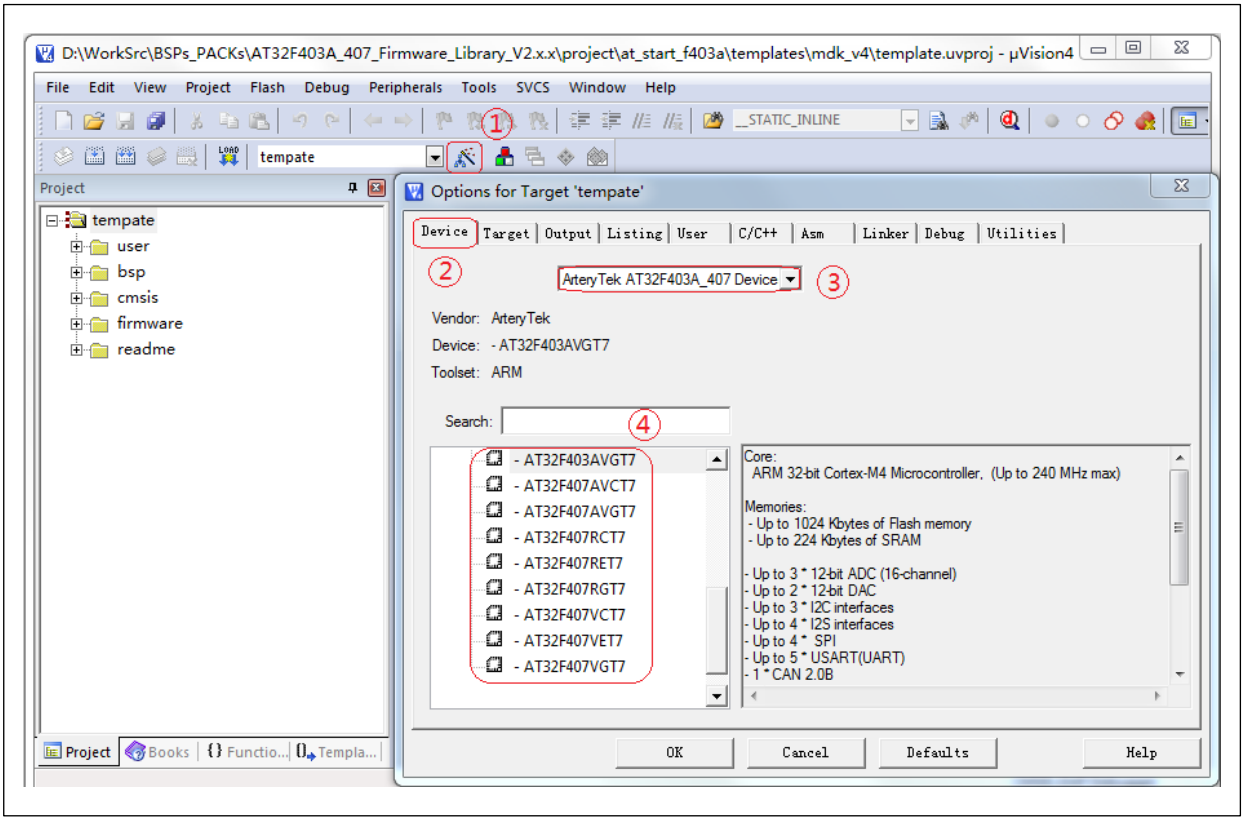
Figure 8. Keil_v4 Pack installation complete



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



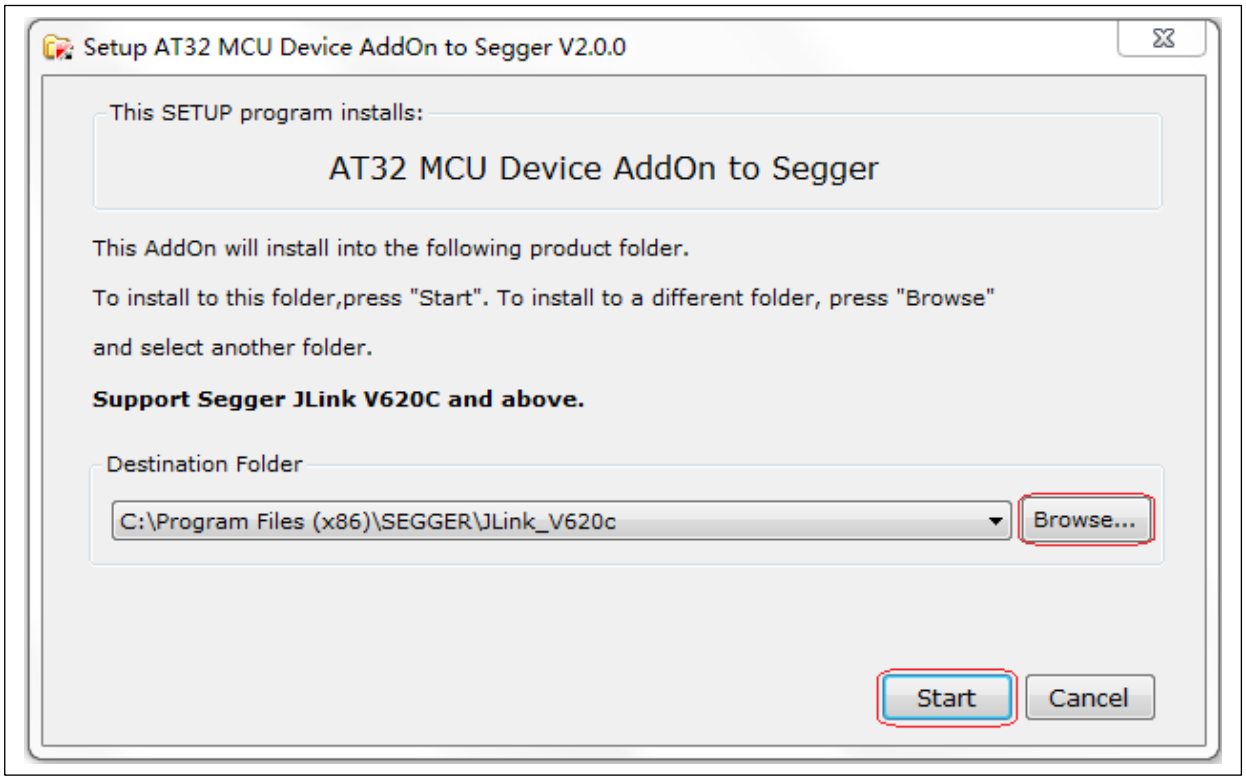
2.4 Segger Pack installation

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

① Unzip Segger_AT32MCU_AddOn.zip

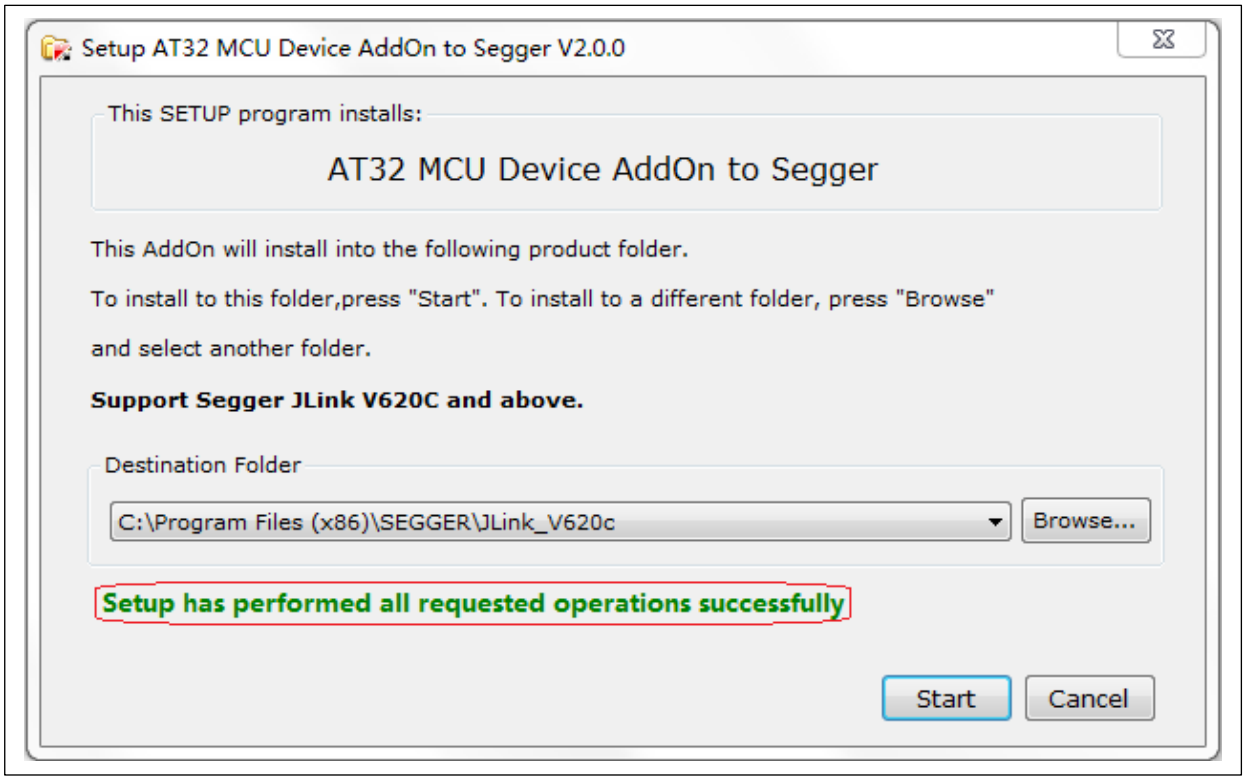
② Double click on Segger_AT32MCU_AddOn.exe, and a dialog box pops up below (the specific version information is subject to the actual conditions)

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.

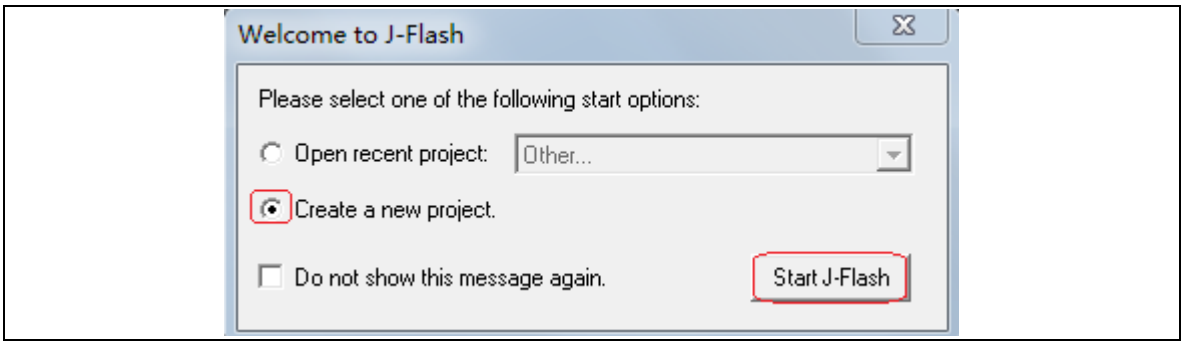
Figure 11. Segger pack installation process



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:

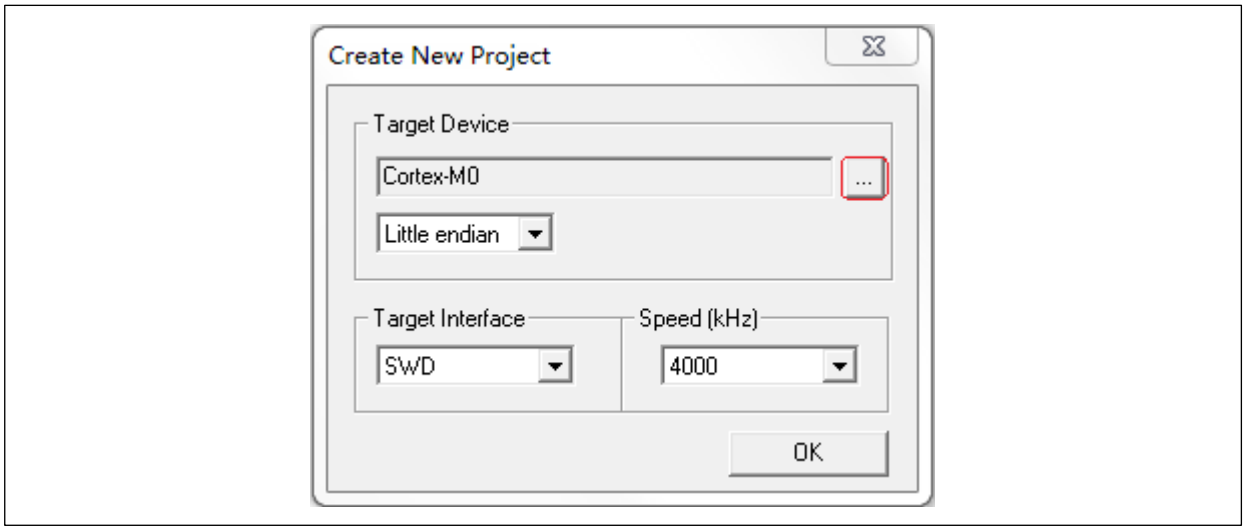
- Open J-Flash.exe, a dialog box appears, tick “Create a new project” and click on “Start J-Flash”:

Figure 12. Open J-Flash



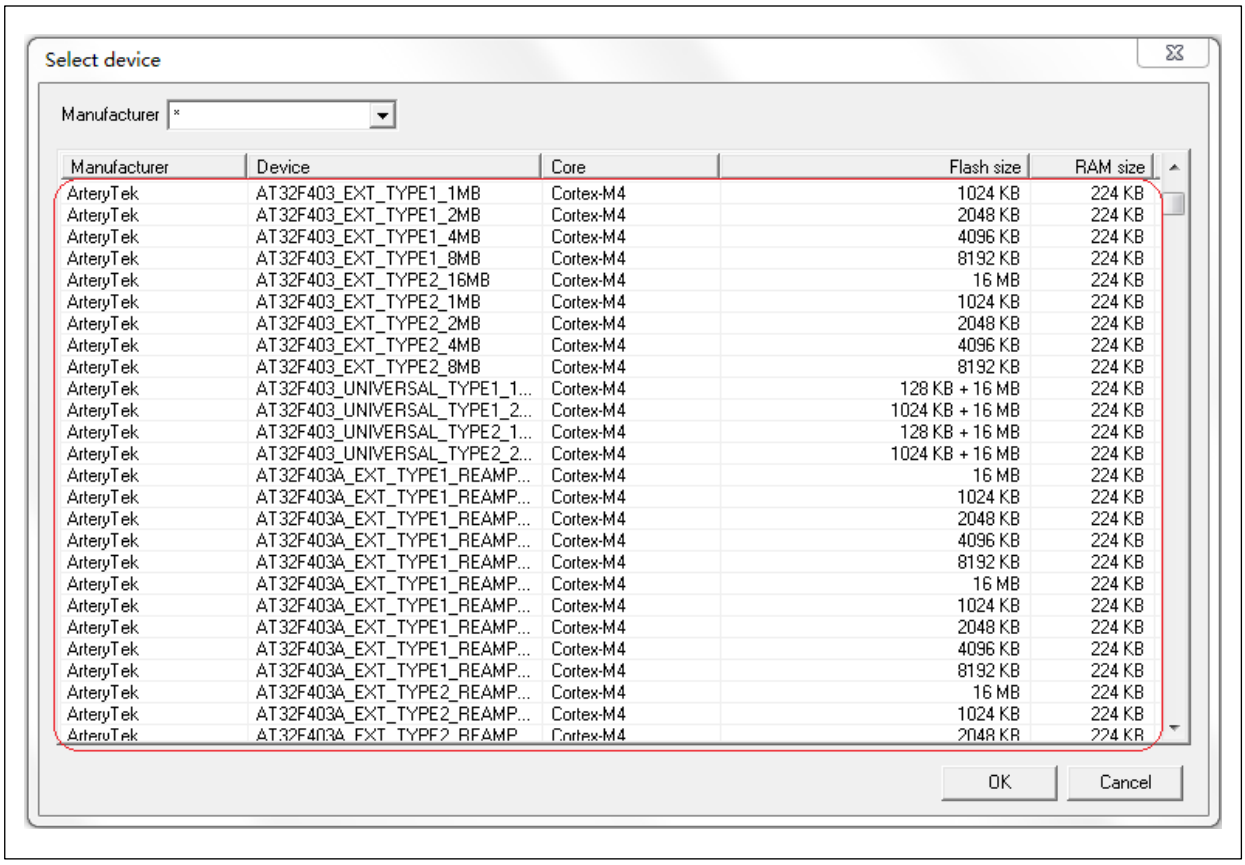
- After “Start J-Flash”, click on the check box under “Target Device”:

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



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.

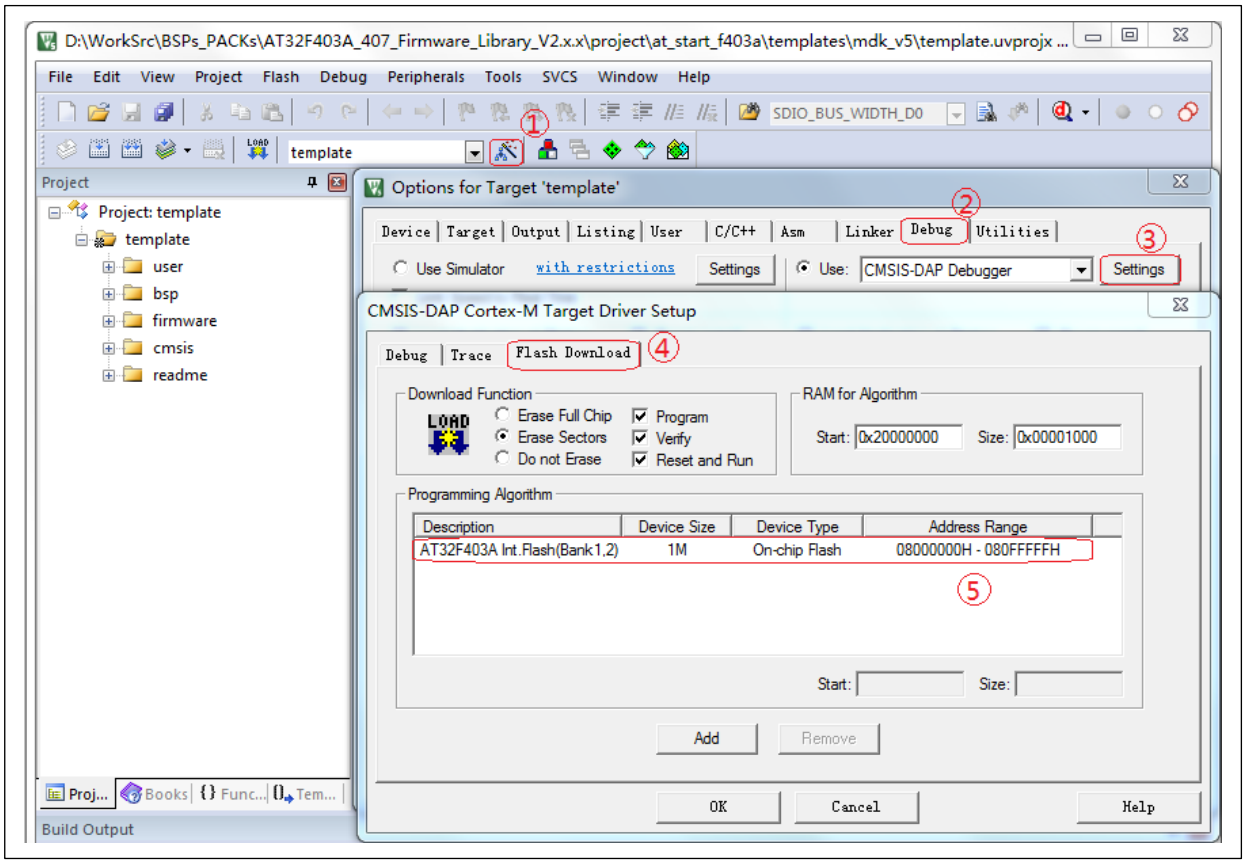
Note: AT32 MUCs have similar Flash algorithms, and this section uses AT32F403A as an example.

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.

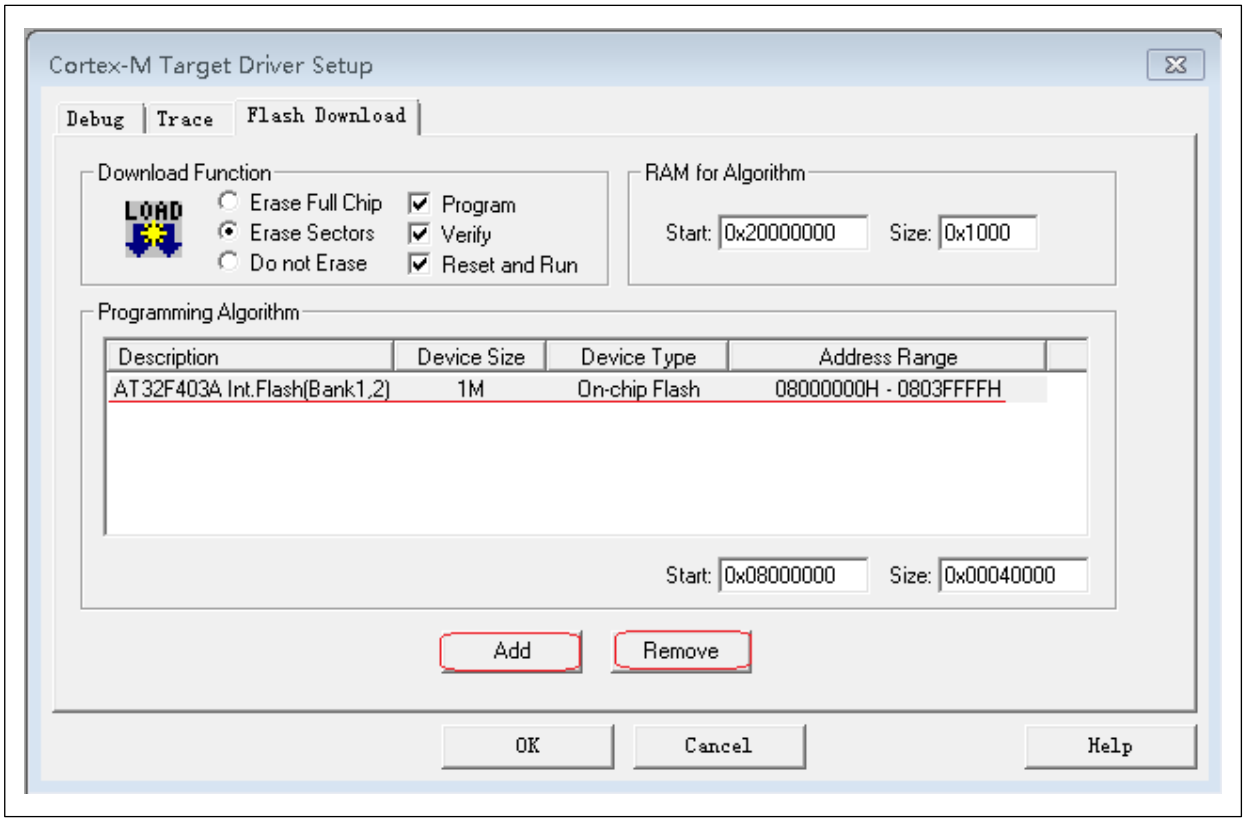
After creating a Keil IDE development tool project, the user can start Debug configuration and select the Flash algorithms. Go to *wand—>Debug—>Settings—>Flash Download*, as shown below:

Figure 15. Keil algorithm file settings



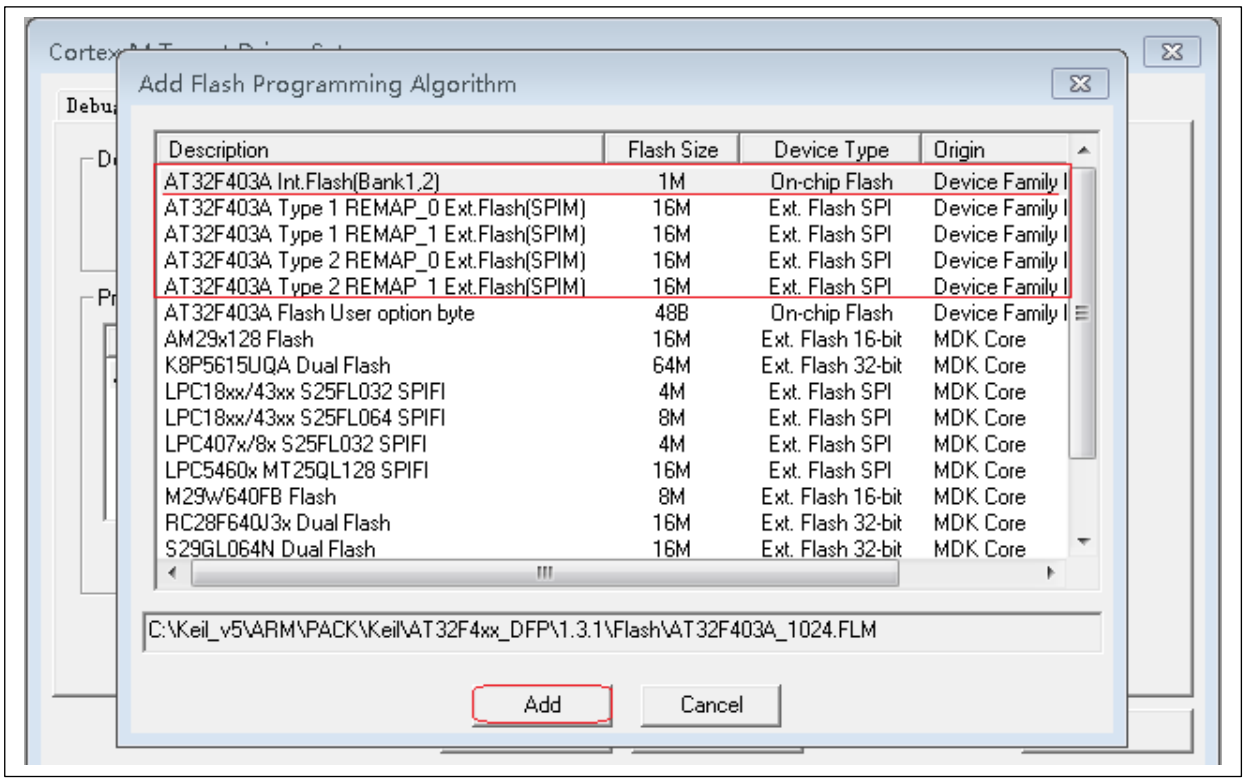
In this example, the selected Flash algorithm file is the default one. To change or remove it, click on this algorithm file, then click on *Add* or *Remove*. If the selected algorithm does not match the MCU, please follow the method below to modify.

Figure 16. Keil algorithm file configuration



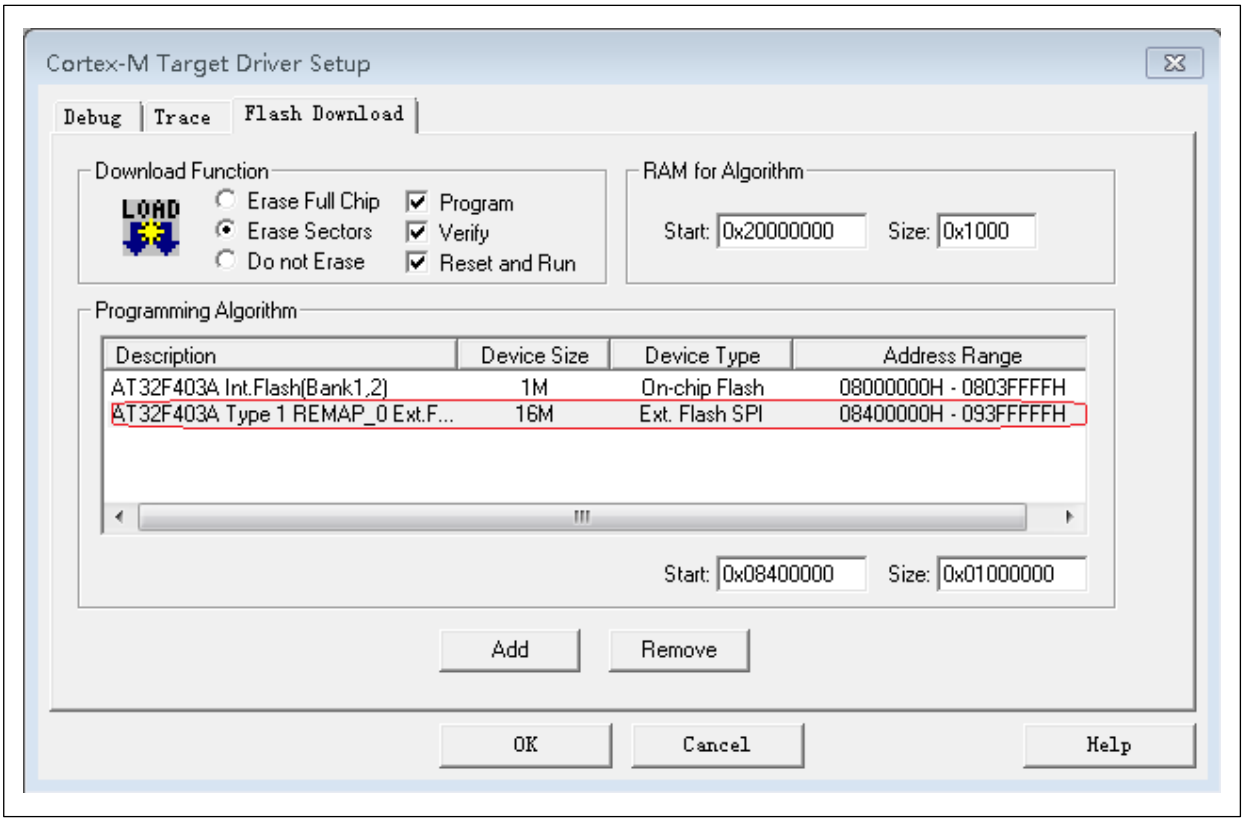
Click on *Remove* to remove the existing algorithm from the configuration, then click on *Add* to view the algorithm files associated with a MCU model and select them, as shown below:

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 SPIM algorithm is added into the project.

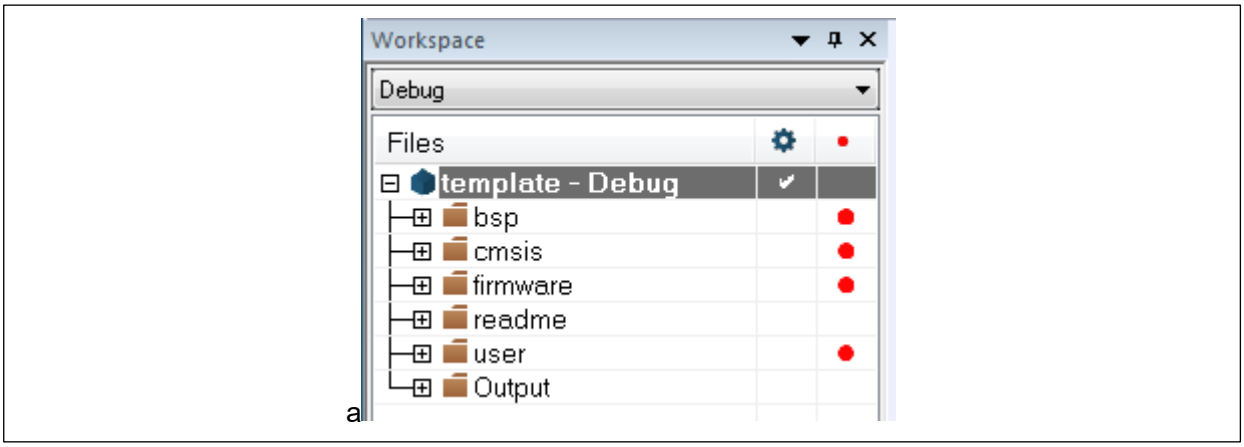
Figure 18. Add algorithm files using Keil



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:

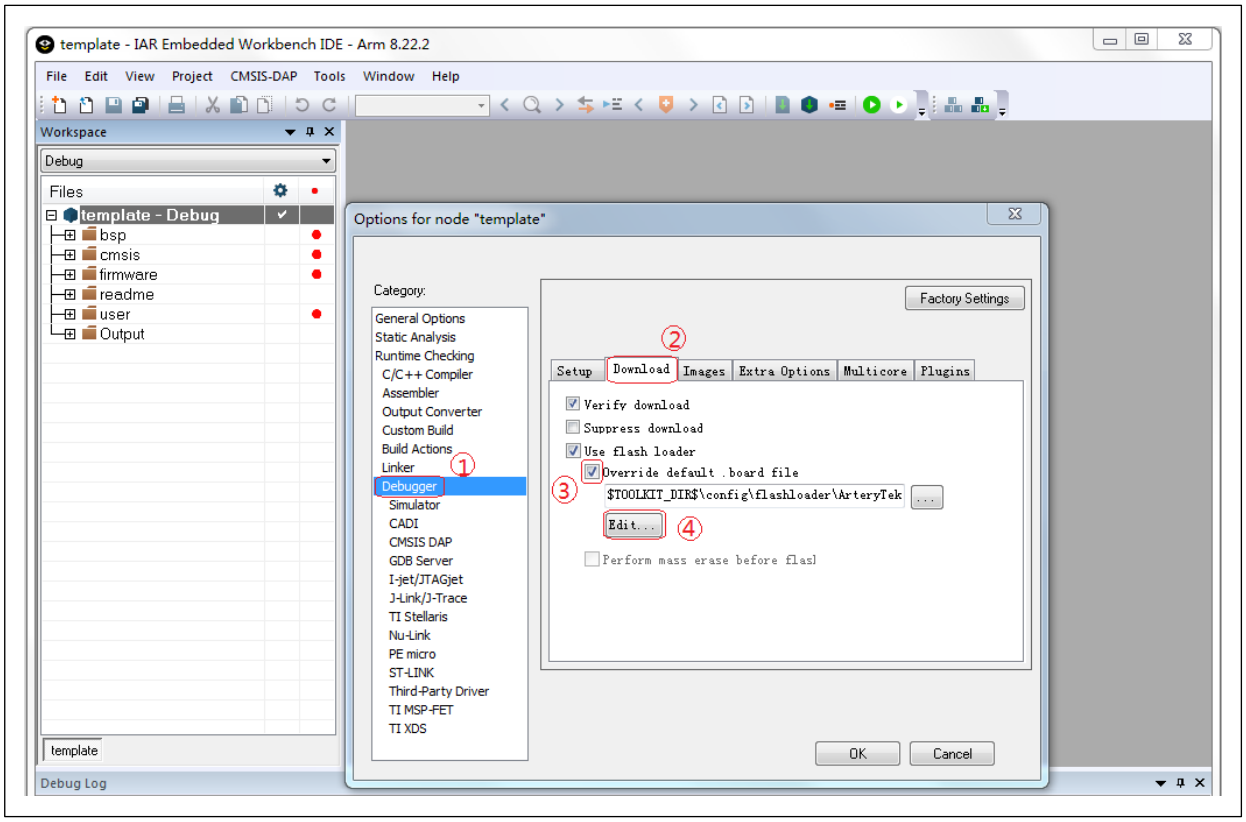
Figure 19. IAR project name



a

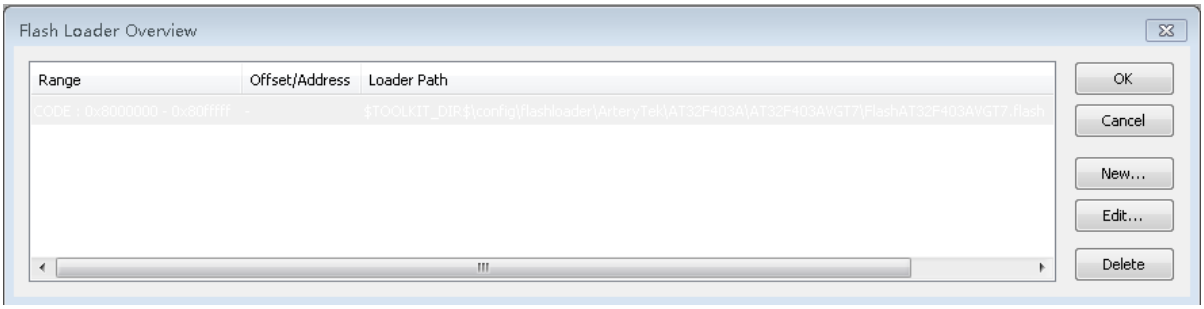
Go to *Options—>Debugger—>Download—>*Tick *Override default .board file—>*Click on *Edit*, as shown below:

Figure 20. IAR algorithm file configuration



Then the following window will be displayed.

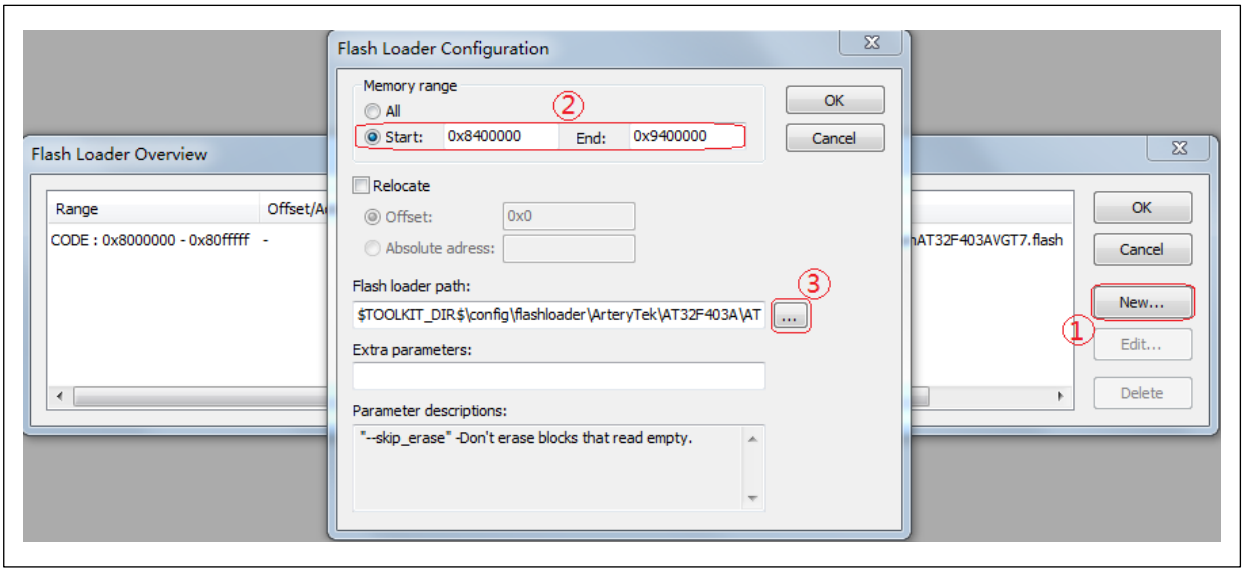
Figure 21. IAR Flash Loader Overview



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

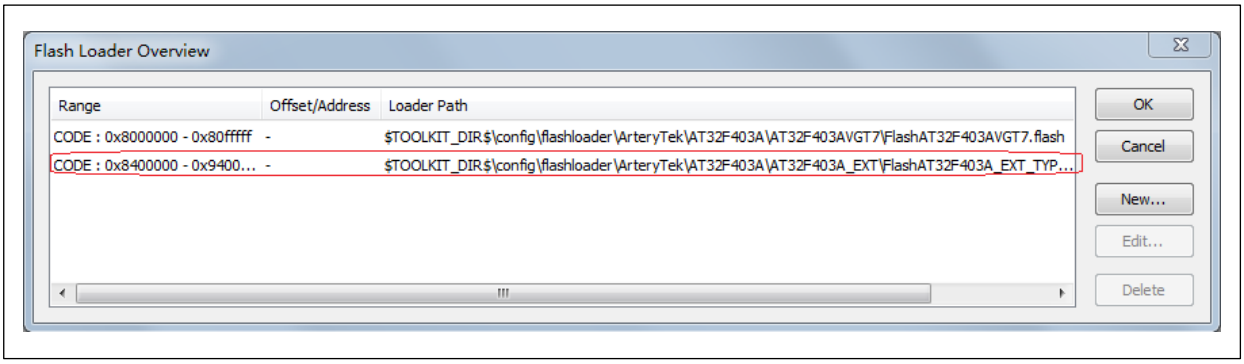
For example, click on *New*—>2. *Memory range*—>3. Select a Flash algorithm file, as shown below:

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

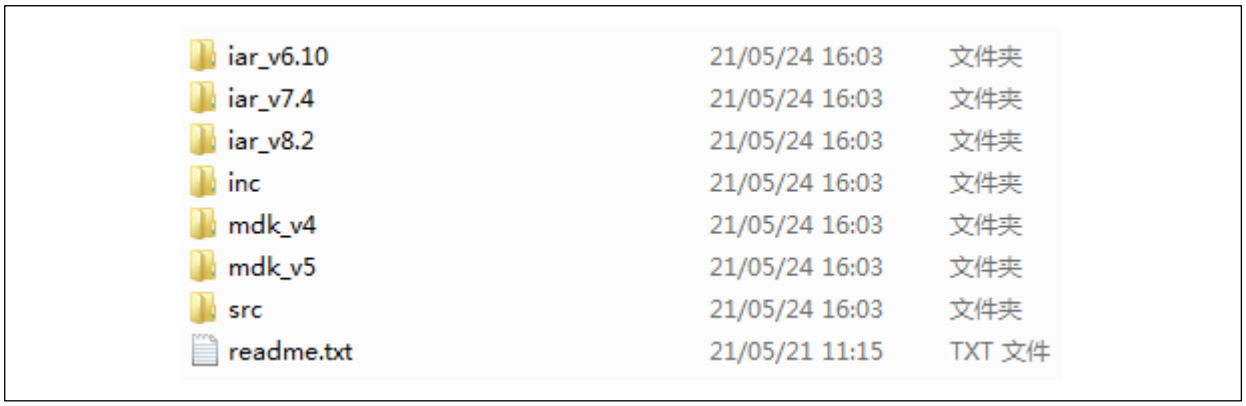
4 BSP introduction

4.1 Quick start

4.1.1 Template project

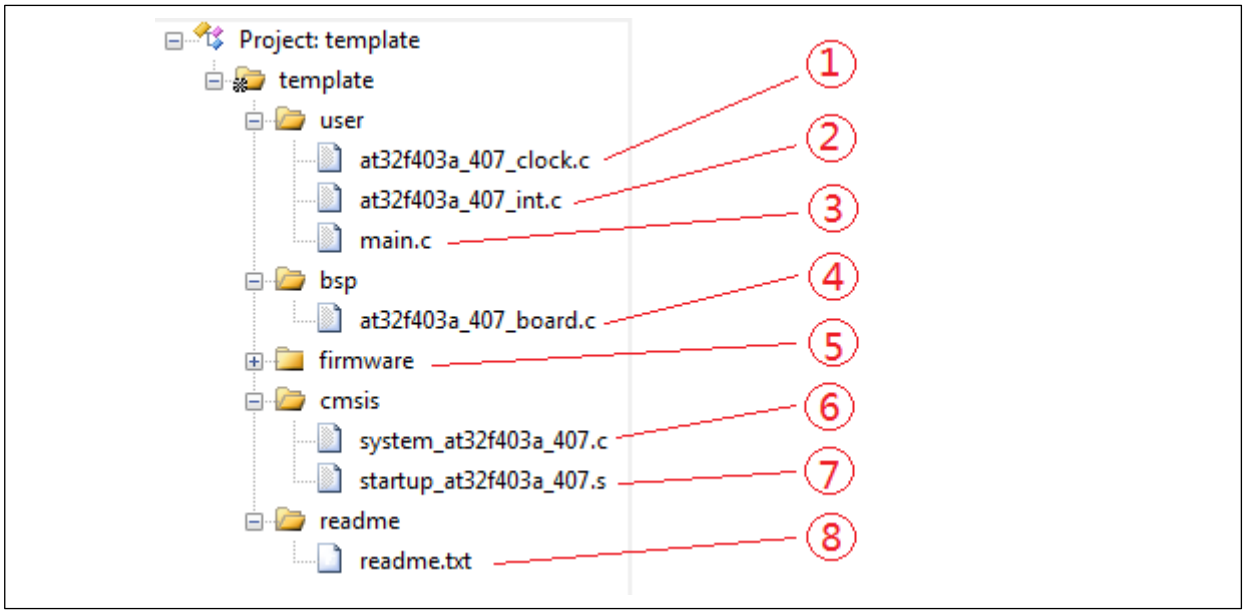
Artery firmware library BSP comes with a series of template projects built around Keil and IAR. For example, the template project of AT32F403A/407 series is located in *AT32F403A_407_Firmware_Library_V2.x.x/project/at_start_xxx/templates*.

Figure 24. Template content



The above template project includes various versions such as Keil_v5, Keil_v4, IAR_6.10, IAR_7.4 and IAR_8.2. Of those, “inc” and “src” folders contain header files and source code files. Open a corresponding folder and click on the corresponding file to open an IDE project. Figure 25 presents an example of Keil_v5 template project (its details and version are subject to the actual firmware library).

Figure 25. Keil_v5 template project example



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

① at32f403a_407_clock.c (clock configuration file) defines the default clock frequency and clock paths

② at32f403a_407_int.c (interrupt file) contains some interrupt handling codes

③ main.c contains the main code files

④ at32f403a_407_board.c (board configuration file) contains common hardware configurations such as buttons and LEDs on the AT-START-Evaluation Board
⑤ at32f403a_407_xx.c under firmware folder contains driver files of on-chip peripherals
⑥ system_at32f403a_407.c is the system initialization file
⑦ startup_at32f403a_407.s is a startup file
⑧ readme.txt is a readme file, containing functional description and configuration information

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

4.1.2 BSP macro definitions

① To create a project, it is necessary to enable a startup code (startup_at32f403a_407.s) and open the appropriate macro definitions according to MCU part number before compiling code. Table 1 presents the correspondence between the MCUs 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**”**, for example, at32f403a_407_adc.h.

- **Constant**

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

  All constants are in written in English capital letters.

- **Variable**

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

- **Naming rules for functions**

  The peripheral functions are named based on the rule of “**peripheral abbreviation_attribute_action**” or “**peripheral abbreviation_action**”.

  The commonly used functions are as follows:

Function type	Naming rule	Example
Peripheral reset	ip_reset	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:

- 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 project, including Keil4, keil5, IAR6, IAR7, IAR8 and eclipse_gcc

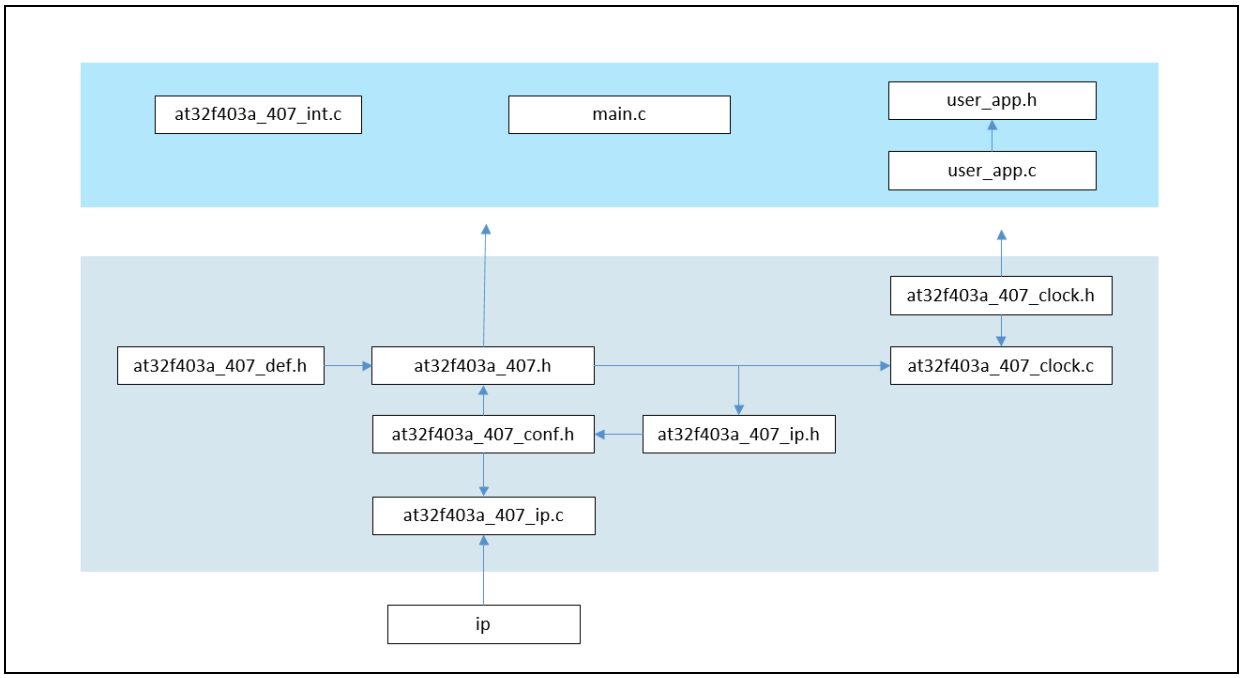
utilities:

Store application cases

4.3.2 BSP function library structure

Figure 28 shows the architecture of BSP function library.

Figure 28. BSP function library structure



BSP function library files are described in Table 3.

Table 3. Summary of BSP function library files

File name	Description
at32f403a_407_conf.h	Macro definition for peripheral enable, and external high-speed clock HEXT_VALUE
main.c	Main function
at32f403a_407_ip.c	Driver source file for a peripheral, for example, at32f403a_407_adc.c
at32f403a_407_ip.h	Driver header file for a peripheral, for example, at32f403a_407_adc.h
at32f403a_407.h	In the header file (at32f403a_407.h), the definition USE_STDPERIPH_DRIVER is used to determine whether the Keil RTE is used or not. Enabling the definition while Keil RTE is unused can prevent Keil-MDK from enabling _RTE_ accidentally.
at32f403a_407_clock.c	This is a clock configuration file used to configure default clock frequency and clock path.
at32f403a_407_clock.h	This is a clock 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 AT32F435/437 peripheral library functions

5.1 HICK automatic clock calibration (ACC)

The ACC register structure acc_type is defined in the “at32f435_437_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 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 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 will increase or decrease by 20 KHz(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 calibration value
Input parameter	NA
Output parameter	NA
Return value	Return acc trimming calibration 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_hicktrim_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 description	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 “at32f435_437_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
ovsp	ADC oversampling register
calval	ADC valibration value register
csts	ADC common status register
cctrl	ADC common control register
codt	ADC common ordinary data register

The table below gives a list of ADC library functions.

Tabl e22. 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 valur for adc_base_struct
adc_base_config	Configure ADC registers with the initialized parameters of the adc_base_struct
adc_common_default_para_init	Define an initial valur for adc_common_struct
adc_common_config	Configure ADC common register with the initialized parameters of the adc_ common _struct
adc_resolution_set	Set ADC conversion resolution
adc_voltage_battery_enable	VBAT enable
adc_dma_mode_enable	Enable DMA transfer for ordinary group
adc_dma_request_repeat_enable	Enable DMA request repetition mode for ordinary group conversion

Function name	Description
adc_interrupt_enable	Enable the selected ADC event interrupt
adc_calibration_value_set	Set calibration value by software
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_conversion_stop	Stop on-going ADC conversion
adc_conversion_stop_status_get	Get the status of stop conversion command
adc_occe_each_conversion_enable	Enable OCCE flag for each ordinary channel conversion
adc_ordinary_software_trigger_enable	Software trigger ordinary group conversion
adc_ordinary_software_trigger_status_get	Get the status of ordinary group conversion triggered by software
adc_preempt_software_trigger_enable	Software trigger preempted group conversion
adc_preempt_software_trigger_status_get	Get the status of preempted group conversion triggered by software
adc_ordinary_conversion_data_get	Get data of ordinary group conversion in non-master-slave mode
adc_combine_ordinary_conversion_data_get	Get data of ordinary group conversion in 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
adc_ordinary_oversample_enable	Enable oversampling of ordinary group
adc_preempt_oversample_enable	Enable oversampling of preempted group
adc_oversample_ratio_shift_set	Set oversampling ratio and its shift length
adc_ordinary_oversample_trig_enable	Enable oversampling trigger mode of ordinary group
adc_ordinary_oversample_restart_set	Enable oversampling restart mode of ordinary group

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	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset()

Example:

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

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 This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	new_state: indicates the pre-configured status of A/D converter This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

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_struc
Input parameter 1	adc_x: indicates the selected ADC peripheral This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_base_struct: adc_base_config_type 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 at32f435_437_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_common_default_para_init function

The table below describes the function adc_common_default_para_init.

Table 27. adc_common_default_para_init function

Name	Description
Function name	adc_common_default_para_init
Function prototype	void adc_common_default_para_init(adc_common_config_type *adc_common_struct)
Function description	Set the initial value for the adc_common_struct
Input parameter	adc_common_struct: adc_common_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The default values of members in the adc_common_struct:

combine_mode: ADC_INDEPENDENT_MODE

div: ADC_HCLK_DIV_2

common_dma_mode: ADC_COMMON_DMAMODE_DISABLE

common_dma_request_repeat_state: FALSE

sampling_interval: ADC_SAMPLING_INTERVAL_5CYCLES

tempervintrv_state: FALSE
vbat_state: FALSE

Example:

```
/* initialize a adc_common_config_type structure */
adc_common_config_type adc_common_struct;
adc_common_default_para_init(&adc_common_struct);
```

5.2.6 adc_common_config function

The table below describes the function adc_common_config.

Table 28. adc_common_config function

Name	Description
Function name	adc_common_config
Function prototype	void adc_common_config(adc_common_config_type *adc_common_struct)
Function description	Initialize ADC common register with the parameters programmed in the adc_common _struct
Input parameter	adc_common_struct: adc_common_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

adc_common_config_type structure

The adc_common_config_type is defined in the at32f435_437_adc.h.

```
typedef struct
{
  adc_combine_mode_type          combine_mode;
  adc_div_type                   div;
  adc_common_dma_mode_type       common_dma_mode;
  confirm_state                  common_dma_request_repeat_state;
  adc_sampling_interval_type     sampling_interval;
  confirm_state                  tempervintrv_state;
  confirm_state                  vbat_state;
} adc_common_config_type; the member parameters are described as follows:
```

combine_mode

Set ADC master-slave combined mode.

ADC_INDEPENDENT_MODE: Non-combined mode

ADC_ORDINARY_SMLT_PREEMPT_SMLT_ONESLAVE_MODE:

Ordinary simultaneous + preempted simultaneous (single slave)

ADC_ORDINARY_SMLT_PREEMPT_INTERLTRIG_ONESLAVE_MODE:

Ordinary simultaneous + alternate preempted trigger (single slave)

ADC_PREEMPT_SMLT_ONLY_ONESLAVE_MODE: Preempted simultaneous (single slave)

ADC_ORDINARY_SMLT_ONLY_ONESLAVE_MODE: Ordinary simultaneous (single slave)

ADC_ORDINARY_SHIFT_ONLY_ONESLAVE_MODE: Ordinary shift (single slave)

ADC_PREEMPT_INTERLTRIG_ONLY_ONESLAVE_MODE:

Alternate preempted trigger (single slave)

ADC_ORDINARY_SMLT_PREEMPT_SMLT_TWOSLAVE_MODE:
Ordinary simultaneous + preempted simultaneous (dual slave)
ADC_ORDINARY_SMLT_PREEMPT_INTERLTRIG_TWOSLAVE_MODE:
Ordinary simultaneous + alternate preempted trigger (dual slave)
ADC_PREEMPT_SMLT_ONLY_TWOSLAVE_MODE: Preempted simultaneous (dual slave)
ADC_ORDINARY_SMLT_ONLY_TWOSLAVE_MODE: Ordinary simultaneous (dual slave)
ADC_ORDINARY_SHIFT_ONLY_TWOSLAVE_MODE: Ordinary shift (dual slave)
ADC_PREEMPT_INTERLTRIG_ONLY_TWOSLAVE_MODE:
Alternate preempted trigger (dual slave)

div

Set ADC frequency division factory.

ADC_HCLK_DIV_2: ADCCLK is clocked by HCLK/2
ADC_HCLK_DIV_3: ADCCLK is clocked by HCLK/3
ADC_HCLK_DIV_4: ADCCLK is clocked by HCLK/4
ADC_HCLK_DIV_5: ADCCLK is clocked by HCLK/5
ADC_HCLK_DIV_6: ADCCLK is clocked by HCLK/6
ADC_HCLK_DIV_7: ADCCLK is clocked by HCLK/7
ADC_HCLK_DIV_8: ADCCLK is clocked by HCLK/8
ADC_HCLK_DIV_9: ADCCLK is clocked by HCLK/9
ADC_HCLK_DIV_10: ADCCLK is clocked by HCLK/10
ADC_HCLK_DIV_11: ADCCLK is clocked by HCLK/11
ADC_HCLK_DIV_12: ADCCLK is clocked by HCLK/12
ADC_HCLK_DIV_13: ADCCLK is clocked by HCLK/13
ADC_HCLK_DIV_14: ADCCLK is clocked by HCLK/14
ADC_HCLK_DIV_15: ADCCLK is clocked by HCLK/15
ADC_HCLK_DIV_16: ADCCLK is clocked by HCLK/16
ADC_HCLK_DIV_17: ADCCLK is clocked by HCLK/17

common_dma_mode

Set ordinary channel DMA transfer mode in master/slave mode.

ADC_COMMON_DMAMODE_DISABLE: No DMA transfer
ADC_COMMON_DMAMODE_1: DMA mode 1
ADC_COMMON_DMAMODE_2: DMA mode 2
ADC_COMMON_DMAMODE_3: DMA mode 3
ADC_COMMON_DMAMODE_4: DMA mode 4
ADC_COMMON_DMAMODE_5: DMA mode 5

common_dma_request_repeat_state

Enable or disable ordinary channel DMA request continuation in master/slave mode.

FALSE: Disabled (after the completion of the programmed number of DMA transfers, no DMA request generated at the end of ordinary conversion)
TRUE: Enabled (don't care about the programmed umber of DMA transfers; each channel sends DMA request at the end of ordinary conversion)

sampling_interval

Set the adjacent ADC sampling interval in ordinary shift mode.

ADC_SAMPLING_INTERVAL_5CYCLES: 5 * TADCCLK
ADC_SAMPLING_INTERVAL_6CYCLES: 6 * TADCCLK
ADC_SAMPLING_INTERVAL_7CYCLES: 7 * TADCCLK

ADC_SAMPLING_INTERVAL_8CYCLES: 8 * TADCCLK
ADC_SAMPLING_INTERVAL_9CYCLES: 9 * TADCCLK
ADC_SAMPLING_INTERVAL_10CYCLES: 10 * TADCCLK
ADC_SAMPLING_INTERVAL_11CYCLES: 11 * TADCCLK
ADC_SAMPLING_INTERVAL_12CYCLES: 12 * TADCCLK
ADC_SAMPLING_INTERVAL_13CYCLES: 13 * TADCCLK
ADC_SAMPLING_INTERVAL_14CYCLES: 14 * TADCCLK
ADC_SAMPLING_INTERVAL_15CYCLES: 15 * TADCCLK
ADC_SAMPLING_INTERVAL_16CYCLES: 16 * TADCCLK
ADC_SAMPLING_INTERVAL_17CYCLES: 17 * TADCCLK
ADC_SAMPLING_INTERVAL_18CYCLES: 18 * TADCCLK
ADC_SAMPLING_INTERVAL_19CYCLES: 19 * TADCCLK
ADC_SAMPLING_INTERVAL_20CYCLES: 20 * TADCCLK

tempervintrv_state

Enable/disable ADC internal temperature sensor and V_{INTRV}.

FALSE: Disable internal temperature sensor and V_{INTRV}

TRUE: Enable internal temperature sensor and V_{INTRV}

vbat_state

Enable/disable ADC V_{BAT}.

FALSE: Disable V_{BAT}

TRUE: Enable V_{BAT}

Example:

```
adc_common_config_type adc_common_struct;
adc_common_struct.combine_mode = ADC_INDEPENDENT_MODE;
adc_common_struct.div = ADC_HCLK_DIV_4;
adc_common_struct.common_dma_mode = ADC_COMMON_DMAMODE_DISABLE;
adc_common_struct.common_dma_request_repeat_state = FALSE;
adc_common_struct.sampling_interval = ADC_SAMPLING_INTERVAL_5CYCLES;
adc_common_struct.tempervintrv_state = TRUE;
adc_common_struct.vbat_state = FALSE;
adc_common_config(&adc_common_struct);
```

5.2.7 adc_resolution_set function

The table below describes the function adc_resolution_set.

Table 29. adc_resolution_set function

Name	Description
Function name	adc_resolution_set
Function prototype	void adc_resolution_set(adc_type *adc_x, adc_resolution_type resolution)
Function description	Set ADC conversion resolution
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	Resolution: ADC conversion resolution selection This parameter can be any enumated value in the adc_resolution_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

resolution

The resolution is used to select ADC conversion resolution, including:

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

Example:

```
/* set conversion resolution */
adc_resolution_set(ADC1, ADC_RESOLUTION_12B);
```

5.2.8 adc_voltage_battery_enable function

The table below describes the function adc_voltage_battery_enable.

Table 30. adc_voltage_battery_enable function

Name	Description
Function name	adc_voltage_battery_enable
Function prototype	void adc_voltage_battery_enable(confirm_state new_state)
Function description	V_{BAT} enable
Input parameter	new_state: indicates the pre-configured status of V_{BAT} enable bit This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable adc voltage battery */
adc_voltage_battery_enable(TRUE);
```

5.2.9 adc_dma_mode_enable function

The table below describes the function adc_dma_mode_enable.

Table 31. 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 This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	new_state: pre-configured status of ordinary group in DMA transfer mode This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.2.10 adc_dma_request_repeat_enable function

The table below describes the function adc_dma_request_repeat_enable

Table 32. adc_dma_request_repeat_enable function

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

Example:

```
/* config dma request repeat,it's not useful when common dma mode is use */
adc_dma_request_repeat_enable(ADC1, TRUE);
```

5.2.11 adc_interrupt_enable function

The table below describes the function adc_interrupt_enable.

Table 33. 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 This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_int: ADC event interrupt selection This parameter is used to select any event interrupt supported by ADC.
Input parameter 3	new_state: indicates the pre-configured status of ADC event interrupts This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

adc_int

The adc_int is used to select and set event interrupts, with the following parameters:

ADC_OCCE_INT: Interrupt enabled at the end of ordinary group conversion

ADC_VMOR_INT: Interrupt enabled when voltage monitor is outside the threshold

ADC_PCCE_INT: Interrpt enabled at the end of preempted group conversion

ADC_OCCO_INT: Overflow interrupt for ordinary group conversion

Example:

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

5.2.12 adc_calibration_value_set

The table below describes the function adc_calibration_value_set.

Table 34. adc_calibration_value_set function

Name	Description
Function name	adc_calibration_value_set
Function prototype	void adc_calibration_value_set(adc_type *adc_x, uint8_t adc_calibration_value)
Function description	Set calibration value by software
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_calibration_value: indicates the pre-configured calibration value This parameter can be any enumerated value in the 0x00~0x7F.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set calibration value */
adc_calibration_value_set(ADC1, 0x3F);
```

5.2.13 adc_calibration_init function

The table below describes the function adc_calibration_init.

Table 35. 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 This parameter can be ADC1, ADC2 or ADC3.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.2.14 adc_calibration_init_status_get function

The table below describes the function adc_calibration_init_status_get.

Table 36. 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 This parameter can be ADC1, ADC2 or ADC3.
Output parameter	NA
Return value	flag_status: indicates the status of calibration initialization Return SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

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

5.2.15 adc_calibration_start function

The table below describes the function adc_calibration_start.

Table 37. 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 This parameter can be ADC1, ADC2 or ADC3.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.2.16 adc_calibration_status_get function

The table below describes the function adc_calibration_status_get.

Table 38. 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 This parameter can be ADC1, ADC2 or ADC3.
Output parameter	NA
Return value	flag_status: indicates the status of calibration Return SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

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

5.2.17 adc_voltage_monitor_enable function

The table below describes the function adc_voltage_monitor_enable.

Table 39. 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 This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_voltage_monitoring: select ordinary group, preempted group or a single channel for voltage monitoring This parameter can be any enumerated value in the adc_voltage_monitoring_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

adc_voltage_monitoring

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);
```

5.2.18 adc_voltage_monitor_threshold_value_set function

The table below describes the function adc_voltage_monitor_threshold_value_set.

Table 40. 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 This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_high_threshold: indicates the upper limit for voltage monitoring This parameter can be any value between 0x000~0xFFFF.
Input parameter 3	adc_low_threshold: indicates the lower limit for voltage monitoring 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);
```

5.2.19 adc_voltage_monitor_single_channel_select function

The table below describes the function adc_voltage_monitor_single_channel_select.

Table 41. adc_voltage_monitor_single_channel_select function

Name	Description
Function name	adc_voltage_monitor_single_channel_select
Function prototype	void adc_voltage_monitor_single_channel_select(adc_type *adc_x, adc_channel_select_type adc_channel)
Function description	Select a single channel for voltage monitoring
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_channel: select a single channel for voltage monitoring Refer to *adc_channel* 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_17: ADC channel 17

ADC_CHANNEL_18: ADC channel 18

Example:

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

5.2.20 adc_ordinary_channel_set function

The table below describes the function adc_ordinary_channel_set.

Table 42. adc_ordinary_channel_set function

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

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

adc_sampletime

The adc_sampletime is used to configure the sampling time of channels, including:

ADC_SAMPLETIME_2_5: sampling time is 2.5 ADCCLK cycles
ADC_SAMPLETIME_6_5: sampling time is 6.5 ADCCLK cycles
ADC_SAMPLETIME_12_5: sampling time is 12.5 ADCCLK cycles
ADC_SAMPLETIME_24_5: sampling time is 24.5 ADCCLK cycles
ADC_SAMPLETIME_47_5: sampling time is 47.5 ADCCLK cycles
ADC_SAMPLETIME_92_5: sampling time is 92.5 ADCCLK cycles
ADC_SAMPLETIME_247_5: sampling time is 247.5 ADCCLK cycles
ADC_SAMPLETIME_640_5: sampling time is 640.5 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_247_5);
adc_ordinary_channel_set(ADC1, ADC_CHANNEL_5, 2, ADC_SAMPLETIME_247_5);
```

5.2.21 adc_preempt_channel_length_set function

The table below describes the function adc_preempt_channel_length_set.

Table 43. 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_lenght)
Function description	Set the length of preempted channel conversion
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_channel_lenght: set the length of preempted channel conversion This parameter can be any value from 0x1~0x4.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.2.22 adc_preempt_channel_set function

The table below describes the function adc_preempt_channel_set

Table 44. 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 This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_channel: indicates the selected channel Refer to *adc_channel* for details.
Input parameter 3	adc_sequence: set the sequence number for channel conversion This parameter can be any value from 1~4.
Input parameter 4	adc_sampletime: set the sampling time for channels Refer to *adc_sampletime* 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_247_5);
adc_preempt_channel_set(ADC1, ADC_CHANNEL_8, 2, ADC_SAMPLETIME_247_5);
```

5.2.23 adc_ordinary_conversion_trigger_set function

The table below describes the function adc_ordinary_conversion_trigger_set.

Table 45. 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, adc_ordinary_trig_edge_type adc_ordinary_trig_edge)
Function description	Enable trigger mode and select trigger events for ordinary group conversion
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_ordinary_trig: indicates the selected trigger event for ordinary group This parameter can be any enumerated value in the adc_ordinary_trig_select_type.
Input parameter 3	adc_ordinary_trig_edge: indicates the pre-configured status of external trigger edge for ordinary group This parameter can be any enumerated value in the adc_ordinary_trig_edge_type.

Name	Description
Output parameter	NA
Return value	NA
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:

ADC_ORDINARY_TRIG_TMR1CH1: TMR1 CH1 event
ADC_ORDINARY_TRIG_TMR1CH2: TMR1 CH2 event
ADC_ORDINARY_TRIG_TMR1CH3: TMR1 CH3 event
ADC_ORDINARY_TRIG_TMR2CH2: TMR2 CH2 event
ADC_ORDINARY_TRIG_TMR2CH3: TMR2 CH3 event
ADC_ORDINARY_TRIG_TMR2CH4: TMR2 CH4 event
ADC_ORDINARY_TRIG_TMR2TRGOUT: TMR2 TRGOUT event
ADC_ORDINARY_TRIG_TMR3CH1: TMR3 CH1 event
ADC_ORDINARY_TRIG_TMR3TRGOUT: TMR3 TRGOUT event
ADC_ORDINARY_TRIG_TMR4CH4: TMR4 CH4 event
ADC_ORDINARY_TRIG_TMR5CH1: TMR5 CH1 event
ADC_ORDINARY_TRIG_TMR5CH2: TMR5 CH2 event
ADC_ORDINARY_TRIG_TMR5CH3: TMR5 CH3 event
ADC_ORDINARY_TRIG_TMR8CH1: TMR8 CH1 event
ADC_ORDINARY_TRIG_TMR8TRGOUT: TMR8 TRGOUT event
ADC_ORDINARY_TRIG_EXINT11: EXINT 11 event
ADC_ORDINARY_TRIG_TMR20TRGOUT: TMR20 TRGOUT event
ADC_ORDINARY_TRIG_TMR20TRGOUT2: TMR20 TRGOUT2 event
ADC_ORDINARY_TRIG_TMR20CH1: TMR20 CH1 event
ADC_ORDINARY_TRIG_TMR20CH2: TMR20 CH2 event
ADC_ORDINARY_TRIG_TMR20CH3: TMR20 CH3 event
ADC_ORDINARY_TRIG_TMR8TRGOUT2: TMR8 TRGOUT2 event
ADC_ORDINARY_TRIG_TMR1TRGOUT2: TMR1 TRGOUT2 event
ADC_ORDINARY_TRIG_TMR4TRGOUT: TMR4 TRGOUT event
ADC_ORDINARY_TRIG_TMR6TRGOUT: TMR6 TRGOUT event
ADC_ORDINARY_TRIG_TMR3CH4: TMR3 CH4 event
ADC_ORDINARY_TRIG_TMR4CH1: TMR4 CH1 event
ADC_ORDINARY_TRIG_TMR1TRGOUT: TMR1 TRGOUT event
ADC_ORDINARY_TRIG_TMR2CH1: TMR2 CH1 event
ADC_ORDINARY_TRIG_TMR7TRGOUT: TMR7 TRGOUT event

adc_ordinary_trig_edge

ADC_ORDINARY_TRIG_EDGE_NONE: Edge trigger disabled
ADC_ORDINARY_TRIG_EDGE_RISING: Rising edge
ADC_ORDINARY_TRIG_EDGE_FALLING: Falling edge
ADC_ORDINARY_TRIG_EDGE_RISING_FALLING: Any edge (rising edge/falling edge)

Example:

```
/* config ordinary trigger source and trigger edge */
adc_ordinary_conversion_trigger_set(ADC1, ADC_ORDINARY_TRIG_TMR1CH1,
ADC_ORDINARY_TRIG_EDGE_NONE);
```

5.2.24 adc_preempt_conversion_trigger_set function

The table below describes the function adc_preempt_conversion_trigger_set.

Table 46. adc_preempt_conversion_trigger_set function

Name	Description
Function name	adc_preempt_conversion_trigger_set
Function prototype	void adc_preempt_conversion_trigger_set(adc_type *adc_x, adc_preempt_trig_select_type adc_preempt_trig, adc_preempt_trig_edge_type adc_preempt_trig_edge)
Function description	Enable trigger mode and trigger events for preempted group conversion
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_preempt_trig: indicates the selected trigger event for preempted group This parameter can be any enumerated value in the adc_preempt_trig_select_type.
Input parameter 3	adc_preempt_trig_edge: indicates the pre-configured status of external trigger edge for preempted group This parameter can be any enumerated value in the adc_preempt_trig_edge_type.
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

ADC_PREEMPT_TRIG_TMR1CH4: TMR1 CH4 event
ADC_PREEMPT_TRIG_TMR1TRGOUT: TMR1 TRGOUT event
ADC_PREEMPT_TRIG_TMR2CH1: TMR2 CH1 event
ADC_PREEMPT_TRIG_TMR2TRGOUT: TMR2 TRGOUT event
ADC_PREEMPT_TRIG_TMR3CH2: TMR3 CH2 event
ADC_PREEMPT_TRIG_TMR3CH4: TMR3 CH4 event
ADC_PREEMPT_TRIG_TMR4CH1: TMR4 CH1 event
ADC_PREEMPT_TRIG_TMR4CH2: TMR4 CH2 event
ADC_PREEMPT_TRIG_TMR4CH3: TMR4 CH3 event
ADC_PREEMPT_TRIG_TMR4TRGOUT: TMR4 TRGOUT event
ADC_PREEMPT_TRIG_TMR5CH4: TMR5 CH4 event
ADC_PREEMPT_TRIG_TMR5TRGOUT: TMR5 TRGOUT event
ADC_PREEMPT_TRIG_TMR8CH2: TMR8 CH2 event
ADC_PREEMPT_TRIG_TMR8CH3: TMR8 CH3 event
ADC_PREEMPT_TRIG_TMR8CH4: TMR8 CH4 event
ADC_PREEMPT_TRIG_EXINT15: EXINT 15 event
ADC_PREEMPT_TRIG_TMR20TRGOUT: TMR20 TRGOUT event
ADC_PREEMPT_TRIG_TMR20TRGOUT2: TMR20 TRGOUT2 event
ADC_PREEMPT_TRIG_TMR20CH4: TMR20 CH4 event
ADC_PREEMPT_TRIG_TMR1TRGOUT2: TMR1 TRGOUT2 event
ADC_PREEMPT_TRIG_TMR8TRGOUT: TMR8 TRGOUT event

ADC_PREEMPT_TRIG_TMR8TRGOUT2: TMR8 TRGOUT2 event
ADC_PREEMPT_TRIG_TMR3CH3: TMR3 CH3 event
ADC_PREEMPT_TRIG_TMR3TRGOUT: TMR3 TRGOUT event
ADC_PREEMPT_TRIG_TMR3CH1: TMR3 CH1 event
ADC_PREEMPT_TRIG_TMR6TRGOUT: TMR6 TRGOUT event
ADC_PREEMPT_TRIG_TMR4CH4: TMR4 CH4 event
ADC_PREEMPT_TRIG_TMR1CH3: TMR1 CH3 event
ADC_PREEMPT_TRIG_TMR20CH2: TMR20 CH2 event
ADC_PREEMPT_TRIG_TMR7TRGOUT: TMR7 TRGOUT event

adc_preempt_trig_edge

ADC_PREEMPT_TRIG_EDGE_NONE: Edge trigger disabled
ADC_PREEMPT_TRIG_EDGE_RISING: Rising edge
ADC_PREEMPT_TRIG_EDGE_FALLING: Falling edge
ADC_PREEMPT_TRIG_EDGE_RISING_FALLING: Any edge (rising edge/falling edge)

Example:

```
/* config preempt trigger source and trigger edge */
adc_preempt_conversion_trigger_set(ADC1, ADC_PREEMPT_TRIG_TMR1CH4,
ADC_PREEMPT_TRIG_EDGE_NONE);
```

5.2.25 adc_preempt_offset_value_set function

The table below describes the function adc_preempt_offset_value_set.

Table 47. adc_preempt_offset_value_set function

Name	Description
Function name	adc_preempt_offset_value_set
Function prototype	void adc_preempt_offset_value_set(adc_type *adc_x, adc_preempt_channel_type adc_preempt_channel, uint16_t adc_offset_value)
Function description	Set the offset value of preempted group conversion
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_preempt_channel: indicates the selected channel Refer to *adc_preempt_channel* for details.
Input parameter 3	adc_offset_value: set the offset value for the selected channel This parameter can be any value from 0x000~0xFFF.
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);
```

5.2.26 adc_ordinary_part_count_set function

The table below describes the function adc_ordinary_part_count_set.

Table 48. adc_ordinary_part_count_set function

Name	Description
Function name	adc_ordinary_part_count_set
Function prototype	void adc_ordinary_part_count_set(adc_type *adc_x, uint8_t adc_channel_count)
Function description	Set the number of ordinary channels at each triggered conversion in partition mode
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_channel_count: indicates the number of ordinary group in partition mode This parameter can be any value from 0x1~0x8.
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 configurable, and that of preempted group is fixed 1.

5.2.27 adc_ordinary_part_mode_enable function

The table below describes the function adc_ordinary_part_mode_enable.

Table 49. adc_ordinary_part_mode_enable function

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

Example:

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

5.2.28 adc_preempt_part_mode_enable function

The table below describes the function adc_preempt_part_mode_enable.

Table 50. 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 This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	new_state: indicates the pre-configured status for partition mode of preempted channels This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.2.29 adc_preempt_auto_mode_enable function

The table below describes the function adc_preempt_auto_mode_enable.

Table 51. adc_preempt _auto_mode_enable function

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

Example:

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

5.2.30 adc_conversion_stop function

The table below describes the function adc_conversion_stop.

Table 52. adc_conversion_stop function

Name	Description
Function name	adc_conversion_stop
Function prototype	void adc_conversion_stop(adc_type *adc_x)
Function description	Stop on-going ADC conversion
Input parameter	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2 or ADC3.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* stop the ongoing conversion */
adc_conversion_stop(ADC1);
```

5.2.31 adc_conversion_stop_status_get function

The table below describes the function adc_conversion_stop_status_get.

Table 53. adc_conversion_stop_status_get function

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

Example:

```
/* wait stop conversion's */
while(adc_conversion_stop_status_get(ADC1));
```

5.2.32 adc_occe_each_conversion_enable function

The table below describes the function adc_occe_each_conversion_enable.

Table 54. adc_occe_each_conversion_enable function

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

Example:

```
/* each ordinary channel conversion set occe flag */
adc_occe_each_conversion_enable(ADC1, TRUE);
```

5.2.33 adc_ordinary_software_trigger_enable function

The table below describes the function adc_ordinary_software_trigger_enable

Table 55. adc_ordinary_software_trigger_enable function

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

Example:

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

5.2.34 adc_ordinary_software_trigger_status_get function

The table below describes the function adc_ordinary_software_trigger_status_get.

Table 56. adc_ordinary_software_trigger_status_get function

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

Example:

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

5.2.35 adc_preempt_software_trigger_enable function

The table below describes the function adc_preempt_software_trigger_enable.

Table 57. 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 This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	new_state: indicates the pre-configured status of software-triggered preempted group conversion This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.2.36 adc_preempt_software_trigger_status_get function

The table below describes the function adc_preempt_software_trigger_status_get.

Table 58. adc_preempt_software_trigger_status_get function

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

Example:

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

5.2.37 adc_ordinary_conversion_data_get function

The table below describes the function adc_ordinary_conversion_data_get.

Table 59. adc_ordinary_conversion_data_get function

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

Example:

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

5.2.38 adc_combine_ordinary_conversion_data_get function

The table below describes the function adc_combine_ordinary_conversion_data_get.

Table 60. adc_combine_ordinary_conversion_data_get function

Name	Description
Function name	adc_combine_ordinary_conversion_data_get
Function prototype	uint32_t adc_combine_ordinary_conversion_data_get(void)
Function description	Get the converted data of ordinary group in master/slave mode
Input parameter	NA
Output parameter	NA
Return value	32-bit converted data by ordinary group
Required preconditions	NA
Called functions	NA

Example:

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

5.2.39 adc_preempt_conversion_data_get function

The table below describes the function adc_preempt_conversion_data_get.

Table 61. adc_preempt_conversion_data_get function

Name	Description
Function name	adc_preempt_conversion_data_get
Function prototype	uint16_t adc_preempt_conversion_data_get(adc_type *adc_x, adc_preempt_channel_type adc_preempt_channel)
Function description	Get the converted data of preempted group
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_preempt_channel: indicates the selected preempted channel Refer to *adc_preempt_channel* 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);
```

5.2.40 adc_flag_get function

The table below describes the function adc_flag_get.

Table 62. adc_flag_get function

Name	Description
Function name	adc_flag_get
Function prototype	flag_status adc_flag_get(adc_type *adc_x, uint8_t adc_flag)
Function description	Get the status of the flag bit
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_flag: indicates the selected flag. Refer to *adc_flag* for details.
Output parameter	NA
Return value	flag_status: the status for the selected flag bit Return 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_OCCE_FLAG: End of ordinary 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
ADC_OCCO_FLAG: Overflow flag for ordinary channel conversion
ADC_RDY_FLAG: ADC ready to conversion flag

Example:

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

5.2.41 adc_interrupt_flag_get function

The table below describes the function adc_interrupt_flag_get.

Table 63. 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 This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_flag: select a flag to be cleard Refer to *adc_flag* 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_OCCE_FLAG: End of ordinary channel conversion

ADC_PCCE_FLAG: End of preempted channel conversion

ADC_PCCS_FLAG: Start of preempted channel conversion

Example:

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

5.2.42 adc_flag_clear function

The table below describes the function adc_flag_clear.

Table 64. adc_flag_clear function

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

Example:

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

5.2.43 adc_ordinary_oversample_enable function

The table below describes the function adc_ordinary_oversample_enable.

Table 65. adc_ordinary_oversample_enable function

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

Name	Description
Called functions	NA

Example:

```
/* enable ordinary oversampling */
adc_ordinary_oversample_enable(ADC1, TRUE);
```

5.2.44 adc_preempt_oversample_enable function

The table below describes the function adc_preempt_oversample_enable.

Table 66. adc_preempt_oversample_enable function

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

Example:

```
/* enable preempt oversampling */
adc_preempt_oversample_enable(ADC1, TRUE);
```

5.2.45 adc_oversample_ratio_shift_set function

The table below describes the function adc_oversample_ratio_shift_set.

Table 67. adc_oversample_ratio_shift_set function

Name	Description
Function name	adc_oversample_ratio_shift_set
Function prototype	void adc_oversample_ratio_shift_set(adc_type *adc_x, adc_oversample_ratio_type adc_oversample_ratio, adc_oversample_shift_type adc_oversample_shift)
Function description	Set oversampling ratio and shift mode
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_oversample_ratio: indicates the oversampling rate This parameter can be any enumerated value in the adc_oversample_ratio_type.
Input parameter 3	adc_oversample_shift: indicates the oversampling shift mode This parameter can be any enumerated value in the adc_oversample_shift_type.
Output parameter	NA

Name	Description
Return value	NA
Required preconditions	NA
Called functions	NA

adc_oversample_ratio

The adc_oversample_ratio is used for ADC oversampling ratio selection, including

ADC_OVERSAMPLE_RATIO_2: 2 x oversampling
ADC_OVERSAMPLE_RATIO_4: 4 x oversampling
ADC_OVERSAMPLE_RATIO_8: 5 x oversampling
ADC_OVERSAMPLE_RATIO_16: 16 x oversampling
ADC_OVERSAMPLE_RATIO_32: 32 x oversampling
ADC_OVERSAMPLE_RATIO_64: 64 x oversampling
ADC_OVERSAMPLE_RATIO_128: 128 x oversampling
ADC_OVERSAMPLE_RATIO_256: 256 x oversampling

adc_oversample_shift

The adc_oversample_shift is used for ADC oversampling bit width selection, including

ADC_OVERSAMPLE_SHIFT_0: No shift
ADC_OVERSAMPLE_SHIFT_1: 1-bit shift
ADC_OVERSAMPLE_SHIFT_2: 2-bit shift
ADC_OVERSAMPLE_SHIFT_3: 3-bit shift
ADC_OVERSAMPLE_SHIFT_4: 4-bit shift
ADC_OVERSAMPLE_SHIFT_5: 5-bit shift
ADC_OVERSAMPLE_SHIFT_6: 6-bit shift
ADC_OVERSAMPLE_SHIFT_7: 7-bit shift
ADC_OVERSAMPLE_SHIFT_8: 8-bit shift

Example:

```
/* set oversampling ratio and shift */
adc_oversample_ratio_shift_set(ADC1, ADC_OVERSAMPLE_RATIO_8, ADC_OVERSAMPLE_SHIFT_3);
```

5.2.46 adc_ordinary_oversample_trig_enable function

The table below describes the function adc_ordinary_oversample_trig_enable.

Table 68. adc_ordinary_oversample_trig_enable function

Name	Description
Function name	adc_ordinary_oversample_trig_enable
Function prototype	void adc_ordinary_oversample_trig_enable(adc_type *adc_x, confirm_state new_state)
Function description	Enable oversampling trigger mode for ordinary group
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	new_state: indicates the pre-configured status of oversampling trigger mode of ordinary group This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA

Name	Description
Required preconditions	NA
Called functions	NA

Example:

```
/* disable ordinary oversampling trigger mode */
adc_ordinary_oversample_trig_enable(ADC1, FALSE);
```

5.2.47 adc_ordinary_oversample_restart_set function

The table below describes the function adc_ordinary_oversample_restart_set.

Table 69. adc_ordinary_oversample_restart_set function

Name	Description
Function name	adc_ordinary_oversample_restart_set
Function prototype	void adc_ordinary_oversample_restart_set(adc_type *adc_x, adc_ordinary_oversample_restart_type adc_ordinary_oversample_restart)
Function description	Select oversampling restart mode for ordinary group
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2 or ADC3.
Input parameter 2	adc_ordinary_oversample_restart: indicates the pre-configurated status of oversampling restart mode for ordinary group This parameter can be any enumerated value in the adc_ordinary_oversample_restart_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

adc_ordinary_oversample_restart

The adc_ordinary_oversample_restart is used to select oversampling restart mode for ordinary group ADC conversion, including:

ADC_OVERSAMPLE_CONTINUE

Continuous mode (the oversampling buffer area of ordinary group is preserved)

ADC_OVERSAMPLE_RESTART

Restart mode (the oversampling buffer area of ordinary group is cleared, that is, the sampling times of the current channel is cleared)

Example:

```
/* set ordinary oversample restart mode */
adc_ordinary_oversample_restart_set(ADC1, ADC_OVERSAMPLE_CONTINUE);
```

5.3 Controller area network (CAN)

CAN register structure can_type is defined in the “at32f435_437_can.h”.

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

The table below gives a list of the CAN registers.

Table 70. Summary of CAN registers

Register	Description
mctrl	CAN master control register
msts	CAN master status register
tsts	CAN transmit status register
rf0	CAN reveive 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
…	…
Fb27f1	CAN filter bank 27 filter register 1
Fb27f2	CAN filter bank 27 filter register 2

The table below gives a list of CAN library functions.

Table 71. 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 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 72. 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 This parameter can be CAN1 or CAN2.
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 73. 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: *can_baudrate_type* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of thecan_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 74. 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 This parameter can be CAN1 or CAN2.
Input parameter 2	can_baudrate_struct: *can_baudrate_type* 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 thecan_baudrate_type before starting.
Called functions	NA

The can_baudrate_type is defined in the at32f435_437_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 75. 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: *can_base_type* 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 76. 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_base_struct: *can_base_type* 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 at32f435_437_can.h.

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

```
  can_msg_discarding_rule_type  mdrsel_selection;
  can_msg_sending_rule_type     mmssr_selection;
} can_base_type;
```

mode_selection

Test mode selection

CAN_MODE_COMMUNICATE: Communication mode

CAN_MODE_LOOPBACK: Loopback mode

CAN_MODE_LISTENONLY: Listen only mode

CAN_MODE_LISTENONLY_LOOPBACK: Loopback + listen only mode

ttc_enable

Enable/disable time-triggered communication mode

FALSE: Disable time-triggered communication mode

TRUE: Enable time-triggered communication mode (while receiving/sending messages, capture time stamp and store it into the CAN_RFCx and CAN_TMCx registers)

aebo_enable

Enable auto exit of bus-off mode

FALSE: Automatic exit of bus-off mode is disabled

TRUE: Automatic exit of bus-off mode is enabled

aed_enable

Enable auto exit of sleep mode

FALSE: Auto exit of sleep mode is disabled

TRUE: Auto exit of sleep mode is enabled

prsf_enable

Disable retransmission when transmit failed

FALSE: Retransmission is enabled

TRUE: Retransmission is disabled

mdrsel_selection

Define message discard rule when reception overflows

CAN_DISCARDING_FIRST_RECEIVED: The previous message is discarded

CAN_DISCARDING_LAST_RECEIVED: The new incoming message is discarded

mmssr_selection

Define multiple message transmit sequence rule

CAN_SENDING_BY_ID: The message with the smallest identifier 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 77. 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: *can_filter_init_type* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of the 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 78. 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 This parameter can be CAN1 or CAN2.
Input parameter 2	can_filter_init_struct: *can_filter_init_type* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of the can_filter_init_type before starting.
Called functions	NA

The can_filter_init_type is defined in the at32f435_437_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: 0~27

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 of the filter identifier 1 (32-bit width, Mask/List mode), 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 identifier 1 (32-bit width, Mask mode), the filter mask identifier 2 (16-bit width, Mask mode), the upper 16 bits of the filter mask identifier 2 (32-bit width, List mode), or the filter mask 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 identifier 1 (32-bit width, Mask mode), the filter mask identifier 2 (16-bit width, Mask mode), the lower 16 bits of the filter identifier 2 (32-bit width, List mode), or the filter mask identifier (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 79. 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 This parameter can be CAN1 or CAN2.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.3.9 can_ttc_mode_enable function

The table below describes the function can_ttc_mode_enable

Table 80. 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 This parameter can be CAN1 or CAN2.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

Note: When the ttc_enable is enabled in the can_base_init, it indicates that only the time stamp is enabled (during message receive and transmit, the time stamp is captured and stored in the CAN_RFCx and CAN_TMCx registers). But when the can_ttc_mode_enable is enabled, not only the time stamp is enable, but the time stamp transmission feature is enabled (during message transmission, the time stamp is sent on the 7th and 8th data byte).

5.3.10 can_message_transmit function

The table below describes the function can_message_transmit.

Table 81. 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 This parameter can be CAN1 or CAN2.
Input parameter 2	tx_message_struct: message pending for transmission, refer to *can_tx_message_type*
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 at32f435_437_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 82. 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 This parameter can be CAN1 or CAN2.
Input parameter 2	transmit_mailbox: indicates the mailbox number required to send messge
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 83. 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 This parameter can be CAN1 or CAN2.
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 84. 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 a frame of message
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1 or CAN2.
Input parameter 2	fifo_number: receive FIFO This parameter can be CAN_RX_FIFO0 or CAN_RX_FIFO1.
Output parameter	rx_message_struct: indicates the received message, refer to *can_rx_message_type*
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 at32f435_437_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: 0x000~0x1FFFFFFF

id_type

Identifier type

CAN_ID_STANDARD: Standard identifier

CAN_ID_EXTENDED: Extended identifier

frame_type

Frame type

CAN_TFT_DATA: Data frame

CAN_TFT_REMOTE: Remote frame

dlc

Data length (in byte)

Value range: 0~8

data[8]

Data pending for transmission

Value range: 0x00~0xFF

filter_index

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

Value range: 0x00~0xFF

Example:

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

5.3.14 can_receive_fifo_release function

The table below describes the function can_receive_fifo_release.

Table 85. 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 This parameter can be CAN1 or CAN2.
Input parameter 2	fifo_number: receive FIFO 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 86. 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 This parameter can be CAN1 or CAN2.
Input parameter 2	fifo_number: receive FIFO 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 87. 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 This parameter can be CAN1 or CAN2.
Input parameter 2	*can_operating_mode*: 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);

/* Start communication: send and receive message*/
…
```

5.3.17 can_doze_mode_enter function

The table below describes the function can_doze_mode_enter.

Table 88. 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 This parameter can be CAN1 or CAN2.
Output parameter	NA
Return value	*can_enter_doze_status*: indicates wheter 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 89. 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 This parameter can be CAN1 or CAN2.
Output parameter	NA
Return value	*can_quit_doze_status*: indicates whethe the Sleep mode has been left
Required preconditions	NA
Called functions	NA

can_quit_doze_status

Indicates whethe 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 90. 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 This parameter can be CAN1 or CAN2.
Output parameter	NA
Return value	*can_error_record*: 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 91. 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 This parameter can be CAN1 or CAN2.
Output parameter	NA
Return value	receive_error_counter: Receive error counter Valur 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 92. 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 This parameter can be CAN1 or CAN2.
Output parameter	NA
Return value	transmit_error_counter: Transmit error counter Value range: 0x00~0xFF
Required preconditions	NA
Called functions	NA

Example:

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

5.3.22 can_interrupt_enable

The table below describes the function can_interrupt_enable.

Table 93. 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 This parameter can be CAN1 or CAN2.
Input parameter 2	*can_int*: Select CAN interrupts
Input parameter 3	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

can_int

CAN interrupt selection

CAN_TCIEN_INT: Transmit mailbox empty interrupt enable
CAN_RF0MIEN_INT: FIFO0 receive message interrupt enable
CAN_RF0FIEN_INT: Receive FIFO0 full interrupt enable
CAN_RF0OIEN_INT: Receive FIFO0 overflow interrupt enable
CAN_RF1MIEN_INT: FIFO1 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 interrrupt */

/* 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 94. 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 This parameter can be CAN1 or CAN2.
Input parameter 2	*can_flag:* indicates the selected flag Refer to the “can_flag” description below for details.
Output parameter	NA
Return value	flag_status: the status of the selected flag Return 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 95. 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 This parameter can be CAN1 or CAN2.
Input parameter 2	*can_flag*: indicates the selected flag Refer to the “can_flag” below for details.
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 non-tempty flag
CAN_RF0FF_FLAG: FIFO0 full flag
CAN_RF0OF_FLAG: FIFO0 overflow flag
CAN_RF1MN_FLAG: FIFO1 non-tempty 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 96. 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 This parameter can be CAN1 or CAN2.
Input parameter 2	*can_flag*: 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_EPF_FLAG: Error passive flag
CAN_BOF_FLAG: Bus-off flag
CAN_ETR_FLAG: Error type record (non-zero Error type flag)
CAN_EOIF_FLAG: Error occur interrupt flag
CAN_TM0TCF_FLAG: Mailbox 0 transmission complete flag
CAN_TM1TCF_FLAG: Mailbox 1 transmission complete flag
CAN_TM2TCF_FLAG: Mailbox 2 transmission complete flag
CAN_RF0FF_FLAG: FIFO0 full flag
CAN_RF0OF_FLAG: FIFO0 overflow flag
CAN_RF1FF_FLAG: FIFO1 full flag
CAN_RF1OF_FLAG: FIFO1 overflow flag
CAN_QDZIF_FLAG: 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 “at32f435_437_crc.h”.

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

} crc_type;
```

The table below gives a list of the CRC registers.

Table 97. Summary of CRC registers

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

The table below gives a list of CRC library functions.

Table 98. Summary of CRC library functions

Function name	Description
crc_data_reset	Data register reset
crc_one_word_calculate	Calulate the CRC value using combination of a new 32-bit data and the previous CRC value
crc_block_calculate	Write a data block in sequence to go through CRC check and return the calculated result
crc_data_get	Get the currently calculated CRC result
crc_common_data_set	Configure common registers
crc_common_date_get	Get the value of common registers
crc_init_data_set	Set the CRC initialization register
crc_reverse_input_data_set	Set CRC input data bit reverse type
crc_reverse_output_data_set	Set CRC output data reverse type
crc_poly_value_set	Set polynomial value
crc_poly_value_get	Get polynomial value
crc_poly_size_set	Set polynomial valid width
crc_poly_size_get	Get polynomial valid width

5.4.1 crc_data_reset function

The table below describes the function crc_data_reset.

Table 99. 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 initiai 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 100. 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 101. 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 102. 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 103. 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 104. 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 common 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 105. crc_init_data_set function

Name	Description
Function name	crc_init_data_set
Function prototype	void crc_init_data_set(uint32_t value);
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 106. crc_reverse_input_data_set function

Name	Description
Function name	crc_reverse_input_data_set
Function prototype	void crc_reverse_input_data_set(crc_reverse_input_type value);
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 107. 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 108. 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 109. crc_poly_value_get function

Name	Description
Function name	crc_poly_value_get
Function prototype	uint32_t crc_poly_value_get(void);
Function description	Get CRC polynomial value
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	uint32_t: 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 110. crc_poly_size_set function

Name	Description
Function name	crc_poly_size_set
Function prototype	void crc_poly_size_set(crc_poly_size_type size);
Function description	Set CRC polynomial valid width
Input parameter 1	size：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 111. 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 “at32f435_437_crm.h”.

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

} crm_type;
```

The table below gives a list of the CRM registers.

Table 112. Summary of CRM registers

Register	Description
ctrl	Clock control register
pllcfg	PLL clock configuration register
cfg	Clock configuration register
clkint	Clock interrupt register
ahbrst1	AHB peripheral reset register 1
ahbrst2	AHB peripheral reset register 2
ahbrst3	AHB peripheral reset register 3
apb1rst	APB1 peripheral reset register
apb2rst	APB2 peripheral reset register
ahben1	AHB peripheral clock enable register 1
ahben2	AHB peripheral clock enable register 2
ahben3	AHB peripheral clock enable register 3
apb1en	APB1 peripheral clock enable register
apb2en	APB2 peripheral clock enable register
ahblpen1	AHB peripheral clock enable in low power mode register 1
ahblpen2	AHB peripheral clock enable in low power mode register 2
ahblpen3	AHB peripheral clock enable in low power mode register 3
apb1lpen	APB1 peripheral clock enable in low power mode register
apb2lpen	APB2 peripheral clock enable in low power mode register
bpdc	Battery powered domain control register
ctrlsts	Control/status register
misc1	Extra register 1
misc2	Extra register 2

The table below gives a list of CRM library functions.

Table 113. Summary of CRM library functions

Function name	Description
crm_reset	Reset clock reset management register and control status
crm_lext_bypass	Configure low-speed external clock bypass
crm_hext_bypass	Configure high-speed external clock bypass
crm_flag_get	Check if the selected flag is set or not
crm_hext_stable_wait	Wait HEXT to get stable
crm_hick_clock_trimming_set	High speed internal clock trimming
crm_hick_clock_calibration_set	High speed internal clock calibration
crm_periph_clock_enable	Peripheral clock enable
crm_periph_reset	Peripheral reset
crm_periph_lowpower_mode_enable	Enable peripheral clock in low-power 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_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	PLL clock source and frequency multiplication factor
crm_sysclk_switch	System clock source switch
crm_sysclk_switch_status_get	Get the status of system clock source
crm_clocks_freq_get	Get clock frequency
crm_clock_out1_set	Select clock source output on clkout1 pin
crm_clock_out2_set	Select clock source output on clkout2 pin
crm_clkout_div_set	Clock frequency division on clock out pins
crm_interrupt_enable	Interrupt enable
crm_auto_step_mode_enable	Auto step-by-step mode 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_to_tmr10_enable	clkout to tmr10 channel 1 enable
crm_emac_output_pulse_set	EMAC output pulse width
crm_pll_parameter_calculate	Calculate PLL parameters automatically

5.5.1 crm_reset function

The table below describes the function crm_reset.

Table 114. 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 115. 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 116. 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 117. 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: indicates 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 118. 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 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 119. 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 120. 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, and 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 121. 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, and its configurable range is from 0 to 0xFF.
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.5.9 crm_periph_clock_enable function

The table below describes the function crm_periph_clock_enable.

Table 122. 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 at32f435_437_crm.h.

The naming rule of this parameter is: CRM_peripheral_PERIPH_CLOCK.

CRM_DMA1_PERIPH_CLOCK: DMA1 peripheral clock

CRM_DMA2_PERIPH_CLOCK: DMA2 peripheral clock

…

CRM_PWC_PERIPH_CLOCK: PWC peripheral clock

CRM_DAC_PERIPH_CLOCK: DAC peripheral clock

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 123. 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 at32f435_437_crm.h.

The naming rule of this parameter is: CRM_peripheral_PERIPH_RESET.

CRM_DMA1_PERIPH_RESET: DMA1 peripheral reset

CRM_DMA2_PERIPH_RESET: DMA 2 peripheral reset

…

CRM_PWC_PERIPH_RESET: PWC peripheral reset

CRM_DAC_PERIPH_RESET: DAC peripheral reset

Example:

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

5.5.11 crm_periph_lowpower_mode_enable function

The table below describes the function crm_periph_lowpower_mode_enable.

Table 124. crm_periph_lowpower_mode_enable function

Name	Description
Function name	crm_periph_lowpower_mode_enable
Function prototype	void crm_periph_lowpower_mode_enable(crm_periph_clock_lowpower_type value, confirm_state new_state);
Function description	Enable peripheral clock in low-power mode
Input parameter 1	value: indicates peripheral clock type in low-power mode
Input parameter 2	new_state: TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

It indicates the selected peripheral.

The crm_periph_lowpower_type is defined in the at32f435_437_crm.h.

The naming rule of this parameter is: CRM_peripheral_PERIPH_LOWPOWER.

CRM_DMA1_PERIPH_LOWPOWER: DMA1 peripheral low-power clock definition

CRM_DMA2_PERIPH_LOWPOWER: DMA2 peripheral low-power clock definition

…

CRM_PWC_PERIPH_LOWPOWER: PWC peripheral low-power clock definition

CRM_DAC_PERIPH_LOWPOWER: DAC peripheral low-power clock definition

Example:

```
/* disable gpioa periph clock at sleep mode */
crm_periph_reset(CRM_GPIOA_PERIPH_LOWPOWER, FALSE);
```

5.5.12 crm_clock_source_enable function

The table below describes the function crm_clock_source_enable.

Table 125. 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 source type
Input parameter 2	new_state: TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

source

Clock source selection

CRM_CLOCK_SOURCE_HICK: HICK

CRM_CLOCK_SOURCE_HEXT: HEXT
CRM_CLOCK_SOURCE_PLL: PLL
CRM_CLOCK_SOURCE_LEXT: LEXT
CRM_CLOCK_SOURCE_LICK: LICK

Example:

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

5.5.13 crm_flag_clear function

The table below describes the function crm_flag_clear.

Table 126. 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.

Table 127. 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_DIV2: HEXT/2 selected as ERTC clock

…

CRM_ERTC_CLOCK_HEXT_DIV31: HEXT/31 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 128. 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 129. 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 130. 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 131. 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_usb_clock_div_set function

The table below describes the function crm_usb_clock_div_set.

Table 132. 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: indicates the 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 used as USB clock

CRM_USB_DIV_1: PLL/1 used as USB clock
CRM_USB_DIV_2_5: PLL/2.5 used as USB clock
CRM_USB_DIV_2: PLL/2 used as USB clock
CRM_USB_DIV_3_5: PLL/3.5 used as USB clock
CRM_USB_DIV_3: PLL/3 used as USB clock
CRM_USB_DIV_4_5: PLL/4.5 used as USB clock
CRM_USB_DIV_4: PLL/4 used as USB clock
CRM_USB_DIV_5_5: PLL/5.5 used as USB clock
CRM_USB_DIV_5: PLL/5 used as USB clock
CRM_USB_DIV_6_5: PLL/6.5 used as USB clock
CRM_USB_DIV_6: PLL/6 used as USB clock
CRM_USB_DIV_7: PLL/7 used as USB clock

Example:

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

5.5.20 crm_clock_failure_detection_enable function

The table below describes the function crm_clock_failure_detection_enable

Table 133. crm_clock_failure_detection_enable function

Name	Description
Function name	crm_clock_failure_detection_enable
Function prototype	void crm_clock_failure_detection_enable(confirm_state new_state);
Function description	Enable clock failure detection
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 clock failure detection */
crm_clock_failure_detection_enable(TRUE);
```

5.5.21 crm_battery_powered_domain_reset function

The table below describes the function crm_battery_powered_domain_reset.

Table 134. crm_battery_powered_domain_reset function

Name	Description
Function name	crm_battery_powered_domain_reset
Function prototype	void crm_battery_powered_domain_reset(confirm_state new_state);
Function description	Reset battery powered domain
Input parameter 1	new_state: Reset (TRUE) or not reset (FALSE)
Input parameter 2	NA
Output parameter	NA

Name	Description
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_battery_powered_domain_reset (TRUE);
```

5.5.22 crm_pll_config function

The table below describes the function crm_pll_config.

Table 135. crm_pll_config function

Name	Description
Function name	crm_pll_config
Function prototype	void crm_pll_config(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 factors
Input parameter 1	clock_source: clock source for PLL frequency multiplication
Input parameter 2	pll_ns: frequency multiplication factor from 31 to 500
Input parameter 3	pll_ms: pre-division frequency factor from 1 to 15
Input parameter 4	pll_fr: post-division frequency factor
Output parameter	NA
Return value	NA
Required preconditions	PLL clock source must be enabled and stable before configuring and enabling PLL.
Called functions	NA

Frequency multiplication formula: PLLCLK = PLL input clock / PLL_MS * PLL_NS / PLL_FR

Requirements:

2MHz <= PLL input clock / PLL_MS <= 16MHz

500MHz <= PLL input clock / PLL_MS * PLL_NS <= 1000MHz

clock_source

CRM_PLL_SOURCE_HICK: HICK is selected as PLL clock source

CRM_PLL_SOURCE_HEXT: 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_config(CRM_PLL_SOURCE_HEXT, 96, 1, CRM_PLL_FR_8);
```

5.5.23 crm_sysclk_switch function

The table below describes the function crm_sysclk_switch.

Table 136. 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 is used as system clock

CRM_SCLK_HEXT: HECT is used as system clock

CRM_SCLK_PLL: PLL is used as system clock

Example:

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

5.5.24 crm_sysclk_switch_status_get function

The table below describes the function crm_sysclk_switch_status_get.

Table 137. 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 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)
{
}
```

5.5.25 crm_clocks_freq_get function

The table below describes the function crm_clocks_freq_get.

Table 138. 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 at32f435_437_crm.h.

```
typedef struct
{
  uint32_t      sclk_freq;
  uint32_t      ahb_freq;
  uint32_t      apb2_freq;
  uint32_t      apb1_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

Example:

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

5.5.26 crm_clock_out1_set function

The table below describes the function crm_clock_out1_set.

Table 139. crm_clock_out1_set function

Name	Description
Function name	crm_clock_out1_set
Function prototype	void crm_clock_out1_set(crm_clkout1_select_type clkout);
Function description	Select clock source output on clkout1 pin
Input parameter 1	clkout: clock source output on clkout1 pin
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

clkout

Select clock source output on the clkout1 pin.

CRM_CLKOUT1_HICK: HICK output on clkout1 pin
CRM_CLKOUT1_HEXT: HEXT output on clkout1 pin
CRM_CLKOUT1_PLL: PLL output on clkout1 pin
CRM_CLKOUT1_LEXT: LEXT output on clkout1 pin

Example:

```
/* config clkout1 output hick */
crm_clock_out1_set(CRM_CLKOUT1_HICK);
```

5.5.27 crm_clock_out2_set function

The table below describes the function crm_clock_out2_set.

Table 140. crm_clock_out2_set function

Name	Description
Function name	crm_clock_out2_set
Function prototype	void crm_clock_out2_set(crm_clkout2_select_type clkout);
Function description	Select clock source output on clkout2 pin
Input parameter 1	clkout: clock source output on clkout2 pin
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

clkout

Select clock source output on the clkout2 pin.

CRM_CLKOUT2_SCLK: SCLK output on clkout2 pin
CRM_CLKOUT2_HEXT: HEXT output on clkout2 pin
CRM_CLKOUT2_PLL: PLL output on clkout2 pin
CRM_CLKOUT2_USB: USB output on clkout2 pin
CRM_CLKOUT2_ADC: ADC output on clkout2 pin
CRM_CLKOUT2_HICK: HICK output on clkout2 pin

CRM_CLKOUT2_LICK: LICK output on clkout2 pin
CRM_CLKOUT2_LEXT: LEXT output on clkout2 pin

Example:

```
/* config clkout2 output hick */
crm_clock_out2_set(CRM_CLKOUT2_SCLK);
```

5.5.28 crm_clkout_div_set function

The table below describes the function crm_clkout_div_set.

Table 141. crm_clkout_div_set function

Name	Description
Function name	crm_clkout_div_set
Function prototype	void crm_clkout_div_set(crm_clkout_index_type index, crm_clkout_div1_type div1, crm_clkout_div2_type div2);
Function description	Clock frequency division on clockout pin
Input parameter 1	index: clkout pin selection
Input parameter 2	div1: divider 1 clock frequency division
Input parameter 3	div2: divider 2 clock frequency division
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

index

clkout pin selection

CRM_CLKOUT_INDEX_1: clkout1
CRM_CLKOUT_INDEX_2: clkout2

div1

Divider 1 clock frequency division

CRM_CLKOUT_DIV1_1: divided by 1
CRM_CLKOUT_DIV1_2: divided by 2
CRM_CLKOUT_DIV1_3: divided by 3
CRM_CLKOUT_DIV1_4: divided by 4
CRM_CLKOUT_DIV1_5: divided by 5

div2

Divider 2 clock frequency division

CRM_CLKOUT_DIV2_1: divided by 1
CRM_CLKOUT_DIV2_2: divided by 2
CRM_CLKOUT_DIV2_4: divided by 4
CRM_CLKOUT_DIV2_8: divided by 8
CRM_CLKOUT_DIV2_16: divided by 16
CRM_CLKOUT_DIV2_64: divided by 64
CRM_CLKOUT_DIV2_128: divided by 128
CRM_CLKOUT_DIV2_256: divided by 256
CRM_CLKOUT_DIV2_512: divided by 512

Example:

```
/* config clkout1 div */
```

```
crm_clkout_div_set (CRM_CLKOUT_INDEX_1, CRM_CLKOUT_DIV1_1, CRM_CLKOUT_DIV2_8);
```

5.5.29 crm_interrupt_enable function

The table below describes the function crm_interrupt_enable.

Table 142. 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 crm interrupt
Input parameter 2	new_state: Enable (TRUE）) or 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 143. 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) or 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 144. 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: 8 MHz HICK used as system clock

CRM_HICK_SCLK_48MHZ: 48 MHz 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 145. 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 HICK (48M) as USB clock source
Input parameter 1	value: PLL or HICK (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_to_tmr10_enable function

The table below describes the function crm_clkout_to_tmr10_enable.

Table 146. crm_clkout_to_tmr10_enable function

Name	Description
Function name	crm_clkout_to_tmr10_enable
Function prototype	void crm_clkout_to_tmr10_enable(confirm_state new_state);
Function description	Enable the clkout to tmr10 channel 1
Input parameter 1	new_state: Enable (TRUE) or disable (FALSE)
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* config clkout internal connect to tmr10 channel1 */
crm_clkout_to_tmr10_enable (TRUE);
```

5.5.34 crm_emac_output_pulse_set function

The table below describes the function crm_emac_output_pulse_set.

Table 147. crm_emac_output_pulse_set function

Name	Description
Function name	crm_emac_output_pulse_set
Function prototype	void crm_emac_output_pulse_set(crm_emac_output_pulse_type width);
Function description	Configure EMAC output pulse width
Input parameter 1	width: emac output pulse width
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

CRM_EMAC_PULSE_125MS: EMAC output pulse width is set as 125ms

CRM_EMAC_PULSE_1SCLK: EMAC output pulse width is set as one system clock cycle

Example:

```
/* config emac output pulse 125ms */
crm_emac_output_pulse_set (CRM_EMAC_PULSE_125MS);
```

5.5.35 crm_pll_parameter_calculate function

The table below describes the function crm_pll_parameter_calculate

Table 148. 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, 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, this parameter can be target_sclk_freq=200000000
Output parameter 1	ret_ms: return pll_ms parameter
Output parameter 2	ret_ns: return pll_ns parameter
Output parameter 3	ret_fr: return pll_fr parameter
Return value	error_status: calculation status. SUCCESS: the calculated result equals the target clock PLL parameter 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 Digital-to-analog converter (DAC)

The DAC register structure dac_type is defined in the at32f435_437_dac.h.

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

The table below gives a list of the DAC registers.

Table 149. Summary of DAC registers

Register	Description
ctrl	DAC control register
swtrg	DAC software trigger register
d1dth12r	DAC1 12-bit right-aligned data holding register
d1dth12l	DAC1 12-bit left-aligned data holding register
d1dth8r	DAC1 8-bit right-aligned data holding register
d2dth12r	DAC2 12-bit right-aligned data holding register
d2dth12l	DAC212-bit left-aligned data holding register
d2dth8r	DAC2 8-bit right-aligned data holding register
ddth12r	Dual DAC 12-bit right-aligned data holding register
ddth12l	Dual DAC 12-bit left-aligned data holding register
ddth8r	Dual DAC 8-bit right-aligned data holding register
d1odt	DAC1 data output register
d2odt	DAC2 data output register

The table below gives a list of DAC library functions

Table 150. Summary of DAC library functions

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

dac_udr_enable	Enable overflow interrupt
dac_udr_interrupt_flag_get	Get DAC1/2 underflow interrupt flag
dac_udr_flag_get	Get DAC1/2 underflow flag
dac_udr_flag_clear	Clear DAC1/2 underflow flag

5.6.1 dac_reset function

The table below describes the function dac_reset.

Table 151. dac_reset function

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

Example:

```
dac_reset ();
```

5.6.2 dac_enable function

The table below describes the function dac_enable.

Table 152. dac_enable function

Name	Description
Function name	dac_enable
Function prototype	void dac_enable(dac_select_type dac_select, confirm_state new_state);
Function description	Enable DAC
Input parameter 1	dac_select: select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dac_enable(DAC1_SELECT, TRUE);
```

5.6.3 dac_output_buffer_enable function

The table below describes the function dac_output_buffer_enable.

Table 153. dac_output_buffer_enable function

Name	Description
Function name	dac_output_buffer_enable
Function prototype	void dac_output_buffer_enable(dac_select_type dac_select, confirm_state new_state);
Function description	Enable DAC output buffer
Input parameter 1	dac_select: select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dac_output_buffer_enable (DAC1_SELECT, TRUE);
```

5.6.4 dac_trigger_enable function

The table below describes the function dac_trigger_enable.

Table 154. dac_trigger_enable function

Name	Description
Function name	dac_trigger_enable
Function prototype	void dac_trigger_enable(dac_select_type dac_select, confirm_state new_state);
Function description	Enable DAC trigger
Input parameter 1	dac_select: select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dac_trigger_enable (DAC1_SELECT, TRUE);
```

5.6.5 dac_trigger_select function

The table below describes the function dac_trigger_select.

Table 155. dac_trigger_select function

Name	Description
Function name	dac_trigger_select
Function prototype	void dac_trigger_select(dac_select_type dac_select, dac_trigger_type dac_trigger_source);
Function description	Select DAC trigger source
Input parameter 1	dac_select: select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Input parameter 2	*dac_trigger_source:* trigger source selected
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dac_trigger_source

Trigger source selection

DAC_TMR6_TRGOUT_EVENT:	TMR6 TRGOUT event triggers DAC
DAC_TMR8_TRGOUT_EVENT:	TMR8 TRGOUT event triggers DAC
DAC_TMR7_TRGOUT_EVENT:	TMR7 TRGOUT event triggers DAC
DAC_TMR5_TRGOUT_EVENT:	TMR5 TRGOUT event triggers DAC
DAC_TMR2_TRGOUT_EVENT:	TMR2 TRGOUT event triggers DAC
DAC_TMR4_TRGOUT_EVENT:	TMR4 TRGOUT event triggers DAC
DAC_EXTERNAL_INTERRUPT_LINE_9:	EXINT LINE 9 event triggers DAC
DAC_SOFTWARE_TRIGGER:	Software triggers DAC

Example:

```
dac_trigger_select(DAC1_SELECT, DAC_TMR2_TRGOUT_EVENT);
dac_trigger_select(DAC2_SELECT, DAC_TMR2_TRGOUT_EVENT);
```

5.6.6 dac_software_trigger_generate function

The table below describes the function dac_software_trigger_generate.

Table 156. dac_software_trigger_generate function

Name	Description
Function name	dac_software_trigger_generate
Function prototype	void dac_software_trigger_generate(dac_select_type dac_select);
Function description	Software triggers DAC
Input parameter 1	dac_select: select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dac_software_trigger_generate (DAC1_SELECT);
```

5.6.7 dac_dual_software_trigger_generate function

The table below describes the function dac_dual_software_trigger_generate.

Table 157. dac_dual_software_trigger_generate function

Name	Description
Function name	dac_dual_software_trigger_generate
Function prototype	void dac_dual_software_trigger_generate(void);
Function description	Trigger DAC1 and DAC2 by software simultaneously
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dac_dual_software_trigger_generate ();
```

5.6.8 dac_wave_generate function

The table below describes the function dac_wave_generate.

Table 158. dac_wave_generate function

Name	Description
Function name	dac_wave_generate
Function prototype	void dac_wave_generate(dac_select_type dac_select, dac_wave_type dac_wave);
Function description	DAC wave generation enable
Input parameter 1	dac_select: select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Input parameter 2	*dac_wave:* wave selected
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dac_wave

Wave generation enable

DAC_WAVE_GENERATE_NONE: Wave generation disabled (output fixed register value)

DAC_WAVE_GENERATE_NOISE: Noise wave generation

DAC_WAVE_GENERATE_TRIANGLE: Triangle wave generation

Example:

```
dac_wave_generate(DAC1_SELECT, DAC_WAVE_GENERATE_NONE);
dac_wave_generate(DAC2_SELECT, DAC_WAVE_GENERATE_NOISE);
```

5.6.9 dac_mask_amplitude_select function

The table below describes the function dac_mask_amplitude_select.

Table 159. dac_mask_amplitude_select function

Name	Description
Function name	dac_mask_amplitude_select
Function prototype	void dac_mask_amplitude_select(dac_select_type dac_select, dac_mask_amplitude_type dac_mask_amplitude);
Function description	DAC noise bit width/triangle amplitude selection
Input parameter 1	dac_select: select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Input parameter 2	*dac_mask_amplitude*: noise bit width/triangle amplitude selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dac_mask_amplitude

Noise bit width/triangle amplitude selection

DAC_LSFR_BIT0_AMPLITUDE_1: LSFR[0] in noise mode/triangle amplitude equal to 1
DAC_LSFR_BIT10_AMPLITUDE_3: LSFR[1:0] in noise mode/triangle amplitude equal to 3
DAC_LSFR_BIT20_AMPLITUDE_7: LSFR[2:0] in noise mode/triangle amplitude equal to 7
DAC_LSFR_BIT30_AMPLITUDE_15: LSFR[3:0] in noise mode/triangle amplitude equal to 15
DAC_LSFR_BIT40_AMPLITUDE_31: LSFR[4:0] in noise mode/triangle amplitude equal to 31
DAC_LSFR_BIT50_AMPLITUDE_63: LSFR[5:0] in noise mode/triangle amplitude equal to 63
DAC_LSFR_BIT60_AMPLITUDE_127: LSFR[6:0] in noise mode/triangle amplitude equal to 127
DAC_LSFR_BIT70_AMPLITUDE_255: LSFR[7:0] in noise mode/triangle amplitude equal to 255
DAC_LSFR_BIT80_AMPLITUDE_511: LSFR[8:0] in noise mode/triangle amplitude equal to 511
DAC_LSFR_BIT90_AMPLITUDE_1023: LSFR[9:0] in noise mode/triangle amplitude equal to 1023
DAC_LSFR_BITA0_AMPLITUDE_2047: LSFR[10:0] in noise mode/triangle amplitude equal to 2047
DAC_LSFR_BITB0_AMPLITUDE_4095: LSFR[11:0] in noise mode/triangle amplitude equal to 4095

Example:

```
dac_mask_amplitude_select (DAC1_SELECT, DAC_LSFR_BIT60_AMPLITUDE_127);
dac_mask_amplitude_select (DAC2_SELECT, DAC_LSFR_BIT80_AMPLITUDE_511);
```

5.6.10 dac_dma_enable function

The table below describes the function dac_dma_enable.

Table 160. dac_dma_enable function

Name	Description
Function name	dac_dma_enable
Function prototype	void dac_dma_enable(dac_select_type dac_select, confirm_state new_state);
Function description	DAC DMA enable
Input parameter 1	dac_select: select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.

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

Example:

```
dac_dma_enable (DAC1_SELECT, TRUE);
```

5.6.11 dac_data_output_get function

The table below describes the function dac_data_output_get.

Table 161. dac_data_output_get function

Name	Description
Function name	dac_data_output_get
Function prototype	uint16_t dac_data_output_get(dac_select_type dac_select);
Function description	Get DAC output value
Input parameter 1	dac_select: select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Output parameter	NA
Return value	dacx_data: dac1/dac2 output value
Required preconditions	NA
Called functions	NA

Example:

```
uint16_t dac1_data;
dac1_data = dac_data_output_get (DAC1_SELECT);
```

5.6.12 dac_1_data_set function

The table below describes the function dac_1_data_set.

Table 162. dac_1_data_set function

Name	Description
Function name	dac_1_data_set
Function prototype	void dac_1_data_set(dac1_aligned_data_type dac1_aligned, uint16_t dac1_data);
Function description	Set DAC1 output data
Input parameter 1	*dac1_aligned*: data format selection
Input parameter 2	*dac1_data*: DAC output value
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dac1_aligned

Data format selection

DAC1_12BIT_RIGHT: 12-bit right-aligned

DAC1_12BIT_LEFT: 12-bit left-aligned

DAC1_8BIT_RIGHT: 8-bit right-aligned

dac1_data

DAC output range settings. Value range varies from one format to another.

DAC1_12BIT_RIGHT: 0x000~0xFFF

DAC1_12BIT_LEFT: 0x0000~0xFFF0

DAC1_8BIT_RIGHT: 0x00~0xFF

Example:

```
dac_1_data_set (DAC1_12BIT_RIGHT, 0x666);
```

5.6.13 dac_2_data_set function

The table below describes the function dac_2_data_set.

Table 163. dac_2_data_set function

Name	Description
Function name	dac_2_data_set
Function prototype	void dac_2_data_set(dac2_aligned_data_type dac2_aligned, uint16_t dac2_data);
Function description	Set DAC2 output value
Input parameter 1	*dac2_aligned*: data format selection
Input parameter 2	*dac2_data*: DAC output value
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dac2_aligned

Data format selection

DAC2_12BIT_RIGHT: 12-bit right-aligned

DAC2_12BIT_LEFT: 12-bit left-aligned

DAC2_8BIT_RIGHT: 8-bit right-aligned

dac2_data

DAC output range settings. Value range varies from one format to another.

DAC2_12BIT_RIGHT: 0x000~0xFFF

DAC2_12BIT_LEFT: 0x0000~0xFFF0

DAC2_8BIT_RIGHT: 0x00~0xFF

Example:

```
dac_2_data_set (DAC2_12BIT_RIGHT, 0x666);
```

5.6.14 dac_dual_data_set function

The table below describes the function dac_dual_data_set.

Table 164. dac_dual_data_set function

Name	Description
Function name	dac_dual_data_set
Function prototype	void dac_dual_data_set(dac_dual_data_type dac_dual, uint16_t data1, uint16_t data2);
Function description	Set DAC1 and DAC2 output values
Input parameter 1	*dac_dual:* data format selection
Input parameter 2	*data1*: DAC1 output value
Input parameter 3	*data2*: DAC2 output value
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dac_dual

Data format selection

DAC_DUAL_12BIT_RIGHT: 12-bit right-aligned

DAC_DUAL_12BIT_LEFT: 12-bit left-aligned

DAC_DUAL_8BIT_RIGHT: 8-bit right-aligned

data1/data2

DAC output range settings. Value range varies from one format to another.

DAC_DUAL_12BIT_RIGHT: 0x000~0xFFF

DAC_DUAL_12BIT_LEFT: 0x0000~0xFFF0

DAC_DUAL_8BIT_RIGHT: 0x00~0xFF

Example:

```
dac_dual_data_set (DAC_DUAL_12BIT_RIGHT, 0x666, 0x777);
```

5.6.15 dac_udr_enable function

The table below describes the function dac_udr_enable.

Table 165. dac_udr_enable function

Name	Description
Function name	dac_udr_enable
Function prototype	void dac_udr_enable(dac_select_type dac_select, confirm_state new_state);
Function description	Enable DAC1/DAC2 underflow interrupt
Input parameter 1	dac_select：select DAC1/2
Input parameter 2	new_state：Enable or disable Return FALSE or TRUE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dac_udr_enable (DAC1_SELECT, TRUE);
```

5.6.16 dac_udr_flag_get function

The table below describes the function dac_udr_flag_get.

Table 166. dac_udr_flag_get function

Name	Description
Function name	dac_udr_flag_get
Function prototype	flag_status dac_udr_flag_get(dac_select_type dac_select);
Function description	Get DAC1/DAC2 overflow flag
Input parameter 1	dac_select：select DAC1/2
Output parameter	NA
Return value	Flag status
Required preconditions	NA
Called functions	NA

Example:

```
dac_udr_flag_get (DAC1_SELECT);
```

5.6.17 dac_udr_interrupt_flag_get function

The table below describes the function dac_udr_interrupt_flag_get.

Table 167. dac_udr_interrupt_flag_get function

Name	Description
Function name	dac_udr_interrupt_flag_get
Function prototype	flag_status dac_udr_interrupt_flag_get (dac_select_type dac_select);
Function description	Get DAC1/ DAC2 overflow interrupt flag
Input parameter 1	dac_select：select DAC1/2
Output parameter	NA
Return value	Flag status
Required preconditions	NA
Called functions	NA

Example:

```
dac_udr_interrupt_flag_get (DAC1_SELECT);
```

5.6.18 dac_udr_flag_clear function

The table below describes the function dac_udr_flag_clear.

Table 168. dac_udr_flag_clear function

Name	Description
Function name	dac_udr_flag_clear
Function prototype	void dac_udr_flag_clear(dac_select_type dac_select);
Function description	Clear DAC1/ DAC2 overflow flag
Input parameter 1	dac_select：select DAC1/2
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dac_udr_flag_clear (DAC1_SELECT);
```

5.7 Debug

The DEBUG register structure debug_type is defined in the “at32f435_437_debug.h”.

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

} debug_type;
```

The table below gives a list of the DEBUG registers.

Table 169. Summary of DEBUG registers

Register	Description
idcode	Device ID
ctrl	Control register

The table below gives a list of DEBUG library functions.

Table 170. Summary of DEBUG library functions

Function name	Description
debug_device_id_get	Read device idcode
debug_periph_mode_set	Peripheral debug mode configuration

5.7.1 debug_device_id_get function

The table below describes the function debug_device_id_get.

Table 171. 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();
```

5.7.2 debug_periph_mode_set function

The table below describes the function debug_periph_mode_set.

Table 172. debug_periph_mode_set function

Name	Description
Function name	debug_periph_mode_set
Function prototype	void debug_periph_mode_set(uint32_t periph_debug_mode, confirm_state new_state);
Function description	Select a peripheral/mode to debug
Input parameter 1	periph_debug_mode: select a peripheral or mode
Input parameter 2	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

periph_debug_mode

Select a peripheral or mode to debug

DEBUG_SLEEP:	Debug in Sleep mode
DEBUG_DEEPSLEEP:	Debug in Deepsleep mode
DEBUG_STANDBY:	Debug in Standby mode
DEBUG_WDT_PAUSE:	Watchdog pause control bit
DEBUG_WWDT_PAUSE:	Window watchdog pause control bit
DEBUG_TMR1_PAUSE:	TMR1 pause control bit
DEBUG_TMR2_PAUSE:	TMR2 pause control bit
DEBUG_TMR3_PAUSE:	TMR3 pause control bit
DEBUG_TMR4_PAUSE:	TMR4 pause control bit

DEBUG_TMR5_PAUSE:	TMR5 pause control bit
DEBUG_TMR6_PAUSE:	TMR6 pause control bit
DEBUG_TMR7_PAUSE:	TMR7 pause control bit
DEBUG_TMR8_PAUSE:	TMR8 pause control bit
DEBUG_TMR9_PAUSE:	TMR9 pause control bit
DEBUG_TMR10_PAUSE:	TMR10 pause control bit
DEBUG_TMR11_PAUSE:	TMR11 pause control bit
DEBUG_TMR12_PAUSE:	TMR12 pause control bit
DEBUG_TMR13_PAUSE:	TMR13 pause control bit
DEBUG_TMR14_PAUSE:	TMR14 pause control bit
DEBUG_TMR20_PAUSE:	TMR20 pause control bit
DEBUG_I2C1_SMBUS_TIMEOUT:	I2C1 SMBUS TIMEOUT pause control bit
DEBUG_I2C2_SMBUS_TIMEOUT:	I2C2 SMBUS TIMEOUT pause control bit
DEBUG_I2C3_SMBUS_TIMEOUT:	I2C3 SMBUS TIMEOUT pause control bit
DEBUG_CAN1_PAUSE:	CAN1 receive register pause control bit
DEBUG_CAN2_PAUSE:	CAN2 receive register pause control bit
DEBUG_ERTC_PAUSE:	ERTC pause control bit
DEBUG_ERTC_512_PAUSE:	ERTC 512Hz pulse output pause control bit

Example:

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

5.8 DMA controller

The DMA register structure dma_type is defined in the “at32f435_437_dma.h”.

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

} dma_type;
```

The DMA channel register structure dma_channel_type is defined in the “at32f435_437_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 173. Summary of DMA registers

Register	Description
dma_sts	DMA status register
dma_clr	DMA status clear register
dma_c1ctrl	DMA channel 1 configuration register
dma_c1dtcnt	DMA channel 1 number of data register
dma_c1paddr	DMA channel 1 peripheral address register
dma_c1maddr	DMA channel 1 memory address register
dma_c2ctrl	DMA channel 2 configuration register
dma_c2dtcnt	DMA channel 2 number of data register
dma_c2paddr	DMA channel 2 peripheral address register
dma_c2maddr	DMA channel 2 memory address register
dma_c3ctrl	DMA channel 3 configuration register
dma_c3dtcnt	DMA channel 3 number of data register
dma_c3paddr	DMA channel 3 peripheral address register
dma_c3maddr	DMA channel 3 memory address register
dma_c4ctrl	DMA channel 4 configuration register
dma_c4dtcnt	DMA channel 4 number of data register
dma_c4paddr	DMA channel 4 peripheral address register
dma_c4maddr	DMA channel 4 memory address register
dma_c5ctrl	DMA channel 5 configuration register
dma_c5dtcnt	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_muxsel	DMAMUX enable register
dma_muxc1ctrl	DMAMUX channel 1 control register
dma_muxc2ctrl	DMAMUX channel 2 control register
dma_muxc3ctrl	DMAMUX channel 3 control register
dma_muxc4ctrl	DMAMUX channel 4 control register
dma_muxc5ctrl	DMAMUX channel 5 control register
dma_muxc6ctrl	DMAMUX channel 6 control register
dma_muxc7ctrl	DMAMUX channel 7 control register
dma_muxg1ctrl	DMAMUX request generator 1 control register
dma_muxg2ctrl	DMAMUX request generator 2 control register
dma_muxg3ctrl	DMAMUX request generator 3 control register
dma_muxg4ctrl	DMAMUX request generator 4 control register
dma_muxsyncsts	DMAMUX synchronous status register
dma_muxsyncclr	DMAMUX synchronous status clear register
dma_muxgsts	DMAMUX request generator status register
dma_muxgclr	DMAMUX request generator status clear register

The table below gives a list of DMA library functions.

Table 174. 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
dmamux_enable	Enable DMAMUX
dmamux_init	Initialize DMAMUX
dmamux_sync_default_para_init	Initialize DMAMUX synchronous module
dmamux_sync_config	Configure DMAMUX synchronous module

dmamux_generator_default_para_init	Initialize DMAMUX request generator
dmamux_generator_config	Configure DMAMUX request generator
dmamux_sync_interrupt_enable	Enable DMAMUX synchronous module interrupt
dmamux_generator_interrupt_enable	Enable DMAMUX request generator interrupt
dmamux_sync_flag_get	Get the flag of DMAMUX synchronous module
dmamux_sync_flag_clear	Clear the flag of DMAMUX synchronous module
dmamux_generator_flag_get	Get the flag of DMAMUX request generator
dmamux_generator_flag_clear	Clear the flag of DMAMUX request generator

5.8.1 dma_default_para_init function

The table below describes the function dma_default_para_init.

Table 175. dma_default_para_init function

Name	Description
Function name	dma_default_para_init
Function prototype	void dma_default_para_init(dma_init_type* dma_init_struct);
Function description	Initialize the parameters of the dma_init_struct
Input parameter 1	dma_init_struct: dma_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of dma_init_struct members.

Table 176. dma_init_struct default values

Member	Default values
peripheral_base_addr	0x0
memory_base_addr	0x0
direction	DMA_DIR_PERIPHERAL_TO_MEMORY
buffer_size	0x0
peripheral_inc_enable	FALSE
memory_inc_enable	FALSE
peripheral_data_width	DMA_PERIPHERAL_DATA_WIDTH_BYTE
memory_data_width	DMA_MEMORY_DATA_WIDTH_BYTE
loop_mode_enable	FALSE
priority	DMA_PRIORITY_LOW

Example:

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

5.8.2 dma_init function

The table below describes the function dma_init.

Table 177. dma_init function

Name	Description
Function name	dma_ init
Function prototype	void dma_init(dma_channel_type* dmax_channely, dma_init_type* dma_init_struct)
Function description	Initialize the selected DMA channel
Input parameter 1	dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1 or 2; y=1...7
Input parameter 2	dma_init_struct: dma_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dma_init_type structure

The dma_init_type is defined in the at32f435_437_dma.h.

```
typedef struct
{
    uint32_t                    peripheral_base_addr;
    uint32_t                    memory_base_addr;
    dma_dir_type                direction;
    uint16_t                    buffer_size;
    confirm_state               peripheral_inc_enable;
    confirm_state               memory_inc_enable;
    dma_peripheral_data_size_type peripheral_data_width;
    dma_memory_data_size_type   memory_data_width;
    confirm_state               loop_mode_enable;
    dma_priority_level_type     priority;
} dma_init_type;
```

peripheral_base_addr

Set the peripheral address of a DMA channel.

memory_base_addr

Set the memory address of a DMA channel.

direction

Set the transfer direction of a DMA channel.

DMA_DIR_PERIPHERAL_TO_MEMORY: Peripheral to memory
DMA_DIR_MEMORY_TO_PERIPHERAL: Memory to peripheral
DMA_DIR_MEMORY_TO_MEMORY: Memory to memory

buffer_size

Set the number of data transfer of a DMA channel.

peripheral_inc_enable

Enable/disable DMA channel peripheral address auto increment.

FALSE: Peripheral address is not incremented
TRUE: Peripheral address is incremented

memory_inc_enable

Enable/disable DMA channel memory address auto increment.

FALSE: Memory address is not incremented

TRUE: Memory address is incremented

peripheral_data_width

Set DMA peripheral data width.

DMA_PERIPHERAL_DATA_WIDTH_BYTE:	Byte
DMA_PERIPHERAL_DATA_WIDTH_HALFWORD:	Half-word
DMA_PERIPHERAL_DATA_WIDTH_WORD:	Word

memory_data_width

Set DMA memory data width.

DMA_MEMORY_DATA_WIDTH_BYTE:	Byte
DMA_MEMORY_DATA_WIDTH_HALFWORD:	Half-word
DMA_MEMORY_DATA_WIDTH_WORD:	Word

loop_mode_enable

Set DMA loop mode.

FALSE: DMA single mode

TRUE: DMA loop mode

priority

Set DMA channel priority.

DMA_PRIORITY_LOW:	Low
DMA_PRIORITY_MEDIUM:	Medium
DMA_PRIORITY_HIGH:	High
DMA_PRIORITY_VERY_HIGH:	Very high

Example:

```
dma_init_type dma_init_struct = {0};
/* dma2 channel1 configuration */
dma_init_struct.buffer_size = BUFFER_SIZE;
dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;
dma_init_struct.memory_base_addr = (uint32_t)src_buffer;
dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_HALFWORD;
dma_init_struct.memory_inc_enable = TRUE;
dma_init_struct.peripheral_base_addr = (uint32_t)0x4001100C;
dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_HALFWORD;
dma_init_struct.peripheral_inc_enable = FALSE;
dma_init_struct.priority = DMA_PRIORITY_MEDIUM;
dma_init_struct.loop_mode_enable = FALSE;
dma_init(DMA2_CHANNEL1, &dma_init_struct);
```

5.8.3 dma_reset function

The table below describes the function dma_reset.

Table 178. dma_reset function

Name	Description
Function name	dma_reset
Function prototype	void dma_reset(dma_channel_type* dmax_channely);
Function description	Reset the selected DMA channel
Input parameter 1	dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1 or 2; y=1...7
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* reset dma2 channel1 */
dma_reset(DMA2_CHANNEL1);
```

5.8.4 dma_data_number_set function

The table below describes the function dma_data_number_set.

Table 179. dma_data_number_set function

Name	Description
Function name	dma_data_number_set
Function prototype	void dma_data_number_set(dma_channel_type* dmax_channely, uint16_t data_number);
Function description	Set the number of data transfer of the selected DMA channel
Input parameter 1	dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1 or 2; y=1...7
Input parameter 2	data_number: indicates the number of data transfer, up to 65535
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set dma2 channel1 data count is 0x100*/
dma_data_number_set(DMA2_CHANNEL1, 0x100);
```

5.8.5 dma_data_number_get function

The table below describes the function dma_data_number_get.

Table 180. dma_data_number_get function

Name	Description
Function name	dma_data_number_get
Function prototype	uint16_t dma_data_number_get(dma_channel_type* dmax_channely);
Function description	Get the number of data transfer of the selected DMA channel
Input parameter 1	dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1 or 2; y=1...7
Output parameter	NA
Return value	Get the number of data transfer of a DMA channel
Required preconditions	NA
Called functions	NA

Example:

```
/* get dma2 channel1 data count*/
uint16_t data_counter;
data_counter = dma_data_number_set(DMA2_CHANNEL1);
```

5.8.6 dma_interrupt_enable function

The table below describes the function dma_interrupt_enable.

Table 181. dma_interrupt_enable function

Name	Description
Function name	dma_interrupt_enable
Function prototype	void dma_interrupt_enable(dma_channel_type* dmax_channely, uint32_t dma_int, confirm_state new_state);
Function description	Enable DMA channel interrupt
Input parameter 1	dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1 or 2; y=1...7
Input parameter 2	dma_int: interrupt source selection
Input parameter 3	new_state: interrupt enable/disable
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dma_int

Select DMA interrupt source

DMA_FDT_INT: Transfer complete interrupt

DMA_HDT_INT: Half transfer complete interrupt

DMA_DTERR_INT: Transfer error interrupt

new_state

Enable or disable DMA channel interrupt

FALSE: Disabled

TRUE: Enabled

Example:

```
/* enable dma2 channel1 transfer full data intterrupt */
dma_interrupt_enable(DMA2_CHANNEL1, DMA_FDT_INT, TRUE);
```

5.8.7 dma_channel_enable function

The table below describes the function dma_channel_enable.

Table 182. 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 or 2; 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(DMA2_CHANNEL1, TRUE);
```

5.8.8 dma_flag_get function

The table below describes the function dma_flag_get.

Table 183. dma_flag_get function

Name	Description
Function name	dma_flag_get
Function prototype	flag_status dma_flag_get(uint32_t dmax_flag);
Function description	Get the flag of the selected DMA channel
Input parameter 1	dmax_flag: select the desired flag
Output parameter	NA
Return value	flag_status: 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
DMA2_GL1_FLAG:	DMA2 channel 1 global flag
DMA2_FDT1_FLAG:	DMA2 channel 1 transfer complete flag
DMA2_HDT1_FLAG:	DMA2 channel 1 half transfer complete flag
DMA2_DTERR1_FLAG:	DMA2 channel 1 transfer error flag
DMA2_GL2_FLAG:	DMA2 channel 2 global flag
DMA2_FDT2_FLAG:	DMA2 channel 2 transfer complete flag
DMA2_HDT2_FLAG:	DMA2 channel 2 half transfer complete flag
DMA2_DTERR2_FLAG:	DMA2 channel 2 transfer error flag
DMA2_GL3_FLAG:	DMA2 channel 3 global flag
DMA2_FDT3_FLAG:	DMA2 channel 3 transfer complete flag
DMA2_HDT3_FLAG:	DMA2 channel 3 half transfer complete flag
DMA2_DTERR3_FLAG:	DMA2 channel 3 transfer error flag
DMA2_GL4_FLAG:	DMA2 channel 4 global flag
DMA2_FDT4_FLAG:	DMA2 channel 4 transfer complete flag
DMA2_HDT4_FLAG:	DMA2 channel 4 half transfer complete flag
DMA2_DTERR4_FLAG:	DMA2 channel 4 transfer error flag
DMA2_GL5_FLAG:	DMA2 channel 5 global flag
DMA2_FDT5_FLAG:	DMA2 channel 5 transfer complete flag
DMA2_HDT5_FLAG:	DMA2 channel 5 half transfer complete flag
DMA2_DTERR5_FLAG:	DMA2 channel 5 transfer error flag

DMA2_GL6_FLAG: DMA2 channel 6 global flag
DMA2_FDT6_FLAG: DMA2 channel 6 transfer complete flag
DMA2_HDT6_FLAG: DMA2 channel 6 half transfer complete flag
DMA2_DTERR6_FLAG: DMA2 channel 6 transfer error flag
DMA2_GL7_FLAG: DMA2 channel 7 global flag
DMA2_FDT7_FLAG: DMA2 channel 7 transfer complete flag
DMA2_HDT7_FLAG: DMA2 channel 7 half transfer complete flag
DMA2_DTERR7_FLAG: DMA2 channel 7 transfer error flag

flag_status

RESET: Flag is reset
SET: Flag is set

Example:

```
if(dma_flag_get(DMA2_FDT1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
}
```

5.8.9 dma_flag_clear function

The table below describes the function dma_flag_clear.

Table 184. dma_flag_clear function

Name	Description
Function name	dma_flag_clear
Function prototype	void dma_flag_clear(uint32_t dmax_flag);
Function description	Clear the selected flag
Input parameter 1	dmax_flag: a flag that needs to be cleared
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dmax_flag

The dmax_flag is used to select the desired flag, including:

DMA1_GL1_FLAG: DMA1 channel 1 global flag
DMA1_FDT1_FLAG: DMA1 channel 1 transfer complete flag
DMA1_HDT1_FLAG: DMA1 channel 1 half transfer complete flag
DMA1_DTERR1_FLAG: DMA1 channel 1 transfer error flag
DMA1_GL2_FLAG: DMA1 channel 2 global flag
DMA1_FDT2_FLAG: DMA1 channel 2 transfer complete flag
DMA1_HDT2_FLAG: DMA1 channel 2 half transfer complete flag
DMA1_DTERR2_FLAG: DMA1 channel 2 transfer error flag
DMA1_GL3_FLAG: DMA1 channel 3 global flag
DMA1_FDT3_FLAG: DMA1 channel 3 transfer complete flag
DMA1_HDT3_FLAG: DMA1 channel 3 half transfer complete flag

DMA1_DTERR3_FLAG:	DMA1 channel 3 transfer error flag
DMA1_GL4_FLAG:	DMA1 channel 4 global flag
DMA1_FDT4_FLAG:	DMA1 channel 4 transfer complete flag
DMA1_HDT4_FLAG:	DMA1 channel 4 half transfer complete flag
DMA1_DTERR4_FLAG:	DMA1 channel 4 transfer error flag
DMA1_GL5_FLAG:	DMA1 channel 5 global flag
DMA1_FDT5_FLAG:	DMA1 channel 5 transfer complete flag
DMA1_HDT5_FLAG:	DMA1 channel 5 half transfer complete flag
DMA1_DTERR5_FLAG:	DMA1 channel 5 transfer error flag
DMA1_GL6_FLAG:	DMA1 channel 6 global flag
DMA1_FDT6_FLAG:	DMA1 channel 6 transfer complete flag
DMA1_HDT6_FLAG:	DMA1 channel 6 half transfer complete flag
DMA1_DTERR6_FLAG:	DMA1 channel 6 transfer error flag
DMA1_GL7_FLAG:	DMA1 channel 7 global flag
DMA1_FDT7_FLAG:	DMA1 channel 7 transfer complete flag
DMA1_HDT7_FLAG:	DMA1 channel 7 half transfer complete flag
DMA1_DTERR7_FLAG:	DMA1 channel 7 transfer error flag
DMA2_GL1_FLAG:	DMA2 channel 1 global flag
DMA2_FDT1_FLAG:	DMA2 channel 1 transfer complete flag
DMA2_HDT1_FLAG:	DMA2 channel 1 half transfer complete flag
DMA2_DTERR1_FLAG:	DMA2 channel 1 transfer error flag
DMA2_GL2_FLAG:	DMA2 channel 2 global flag
DMA2_FDT2_FLAG:	DMA2 channel 2 transfer complete flag
DMA2_HDT2_FLAG:	DMA2 channel 2 half transfer complete flag
DMA2_DTERR2_FLAG:	DMA2 channel 2 transfer error flag
DMA2_GL3_FLAG:	DMA2 channel 3 global flag
DMA2_FDT3_FLAG:	DMA2 channel 3 transfer complete flag
DMA2_HDT3_FLAG:	DMA2 channel 3 half transfer complete flag
DMA2_DTERR3_FLAG:	DMA2 channel 3 transfer error flag
DMA2_GL4_FLAG:	DMA2 channel 4 global flag
DMA2_FDT4_FLAG:	DMA2 channel 4 transfer complete flag
DMA2_HDT4_FLAG:	DMA2 channel 4 half transfer complete flag
DMA2_DTERR4_FLAG:	DMA2 channel 4 transfer error flag
DMA2_GL5_FLAG:	DMA2 channel 5 global flag
DMA2_FDT5_FLAG:	DMA2 channel 5 transfer complete flag
DMA2_HDT5_FLAG:	DMA2 channel 5 half transfer complete flag
DMA2_DTERR5_FLAG:	DMA2 channel 5 transfer error flag
DMA2_GL6_FLAG:	DMA2 channel 6 global flag
DMA2_FDT6_FLAG:	DMA2 channel 6 transfer complete flag
DMA2_HDT6_FLAG:	DMA2 channel 6 half transfer complete flag
DMA2_DTERR6_FLAG:	DMA2 channel 6 transfer error flag
DMA2_GL7_FLAG:	DMA2 channel 7 global flag
DMA2_FDT7_FLAG:	DMA2 channel 7 transfer complete flag
DMA2_HDT7_FLAG:	DMA2 channel 7 half transfer complete flag
DMA2_DTERR7_FLAG:	DMA2 channel 7 transfer error flag

Example:

```
if(dma_flag_get(DMA2_FDT1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
    dma_flag_clear(DMA2_FDT1_FLAG);
}
```

5.8.10 dma_flexible_config function

The table below describes the function dma_flexible_enable.

Table 185. dma_flexible_config function

Name	Description
Function name	dma_flexible_config
Function prototype	void dma_flexible_config(dma_type* dma_x, dmamux_channel_type *dmamux_channelx, dmamux_requst_id_sel_type dmamux_req_sel);
Function description	Configure DMAMUX
Input parameter 1	dma_x: DMAx, x=1 or 2
Input parameter 2	*dmamux_channelx* :DMAMUX channel selected, x=1…7
Input parameter 3	*dmamux_req_sel* :DMAMUX channel request ID
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dmamux_channelx

DMAMUX channel selection, including:

DMA1MUX_CHANNEL1

DMA1MUX_CHANNEL2

DMA1MUX_CHANNEL3

DMA1MUX_CHANNEL4

DMA1MUX_CHANNEL5

DMA1MUX_CHANNEL6

DMA1MUX_CHANNEL7

DMA2MUX_CHANNEL1

DMA2MUX_CHANNEL2

DMA2MUX_CHANNEL3

DMA2MUX_CHANNEL4

DMA2MUX_CHANNEL5

DMA2MUX_CHANNEL6

DMA2MUX_CHANNEL7

dmamux_req_sel

The table below shows the DMAMUX channel request ID.

Table 186. DMAMUX channel request source ID

Request source ID	Description	Request source ID	Description
0x01	DMAMUX_DMAREQ_ID_REQ_G1	0x02	DMAMUX_DMAREQ_ID_REQ_G2
0x03	DMAMUX_DMAREQ_ID_REQ_G3	0x04	DMAMUX_DMAREQ_ID_REQ_G4
0x05	DMAMUX_DMAREQ_ID_ADC1	0x24	DMAMUX_DMAREQ_ID_ADC2
0x25	DMAMUX_DMAREQ_ID_ADC3	0x06	DMAMUX_DMAREQ_ID_DAC1
0x29	DMAMUX_DMAREQ_ID_DAC2	0x08	DMAMUX_DMAREQ_ID_TMR6_OVERFLOW
0x09	DMAMUX_DMAREQ_ID_TMR7_OVERFLOW	0x0A	DMAMUX_DMAREQ_ID_SPI1_RX
0x0B	DMAMUX_DMAREQ_ID_SPI1_TX	0x0C	DMAMUX_DMAREQ_ID_SPI2_RX
0x0D	DMAMUX_DMAREQ_ID_SPI2_TX	0x0E	DMAMUX_DMAREQ_ID_SPI3_RX
0x0F	DMAMUX_DMAREQ_ID_SPI3_TX	0x6A	DMAMUX_DMAREQ_ID_SPI4_RX
0x6B	DMAMUX_DMAREQ_ID_SPI4_TX	0x6E	DMAMUX_DMAREQ_ID_I2S2_EXT_RX
0x6F	DMAMUX_DMAREQ_ID_I2S2_EXT_TX	0x70	DMAMUX_DMAREQ_ID_I2S3_EXT_RX
0x71	DMAMUX_DMAREQ_ID_I2S3_EXT_TX	0x10	DMAMUX_DMAREQ_ID_I2C1_RX
0x11	DMAMUX_DMAREQ_ID_I2C1_TX	0x12	DMAMUX_DMAREQ_ID_I2C2_RX
0x13	DMAMUX_DMAREQ_ID_I2C2_TX	0x14	DMAMUX_DMAREQ_ID_I2C3_RX
0x15	DMAMUX_DMAREQ_ID_I2C3_TX	0x18	DMAMUX_DMAREQ_ID_USART1_RX
0x19	DMAMUX_DMAREQ_ID_USART1_TX	0x1A	DMAMUX_DMAREQ_ID_USART2_RX
0x1B	DMAMUX_DMAREQ_ID_USART2_TX	0x1C	DMAMUX_DMAREQ_ID_USART3_RX
0x1D	DMAMUX_DMAREQ_ID_USART3_TX	0x1E	DMAMUX_DMAREQ_ID_UART4_RX
0x1F	DMAMUX_DMAREQ_ID_UART4_TX	0x20	DMAMUX_DMAREQ_ID_UART5_RX
0x21	DMAMUX_DMAREQ_ID_UART5_TX	0x72	DMAMUX_DMAREQ_ID_USART6_RX
0x73	DMAMUX_DMAREQ_ID_USART6_TX	0x74	DMAMUX_DMAREQ_ID_UART7_RX
0x75	DMAMUX_DMAREQ_ID_UART7_TX	0x76	DMAMUX_DMAREQ_ID_UART8_RX
0x77	DMAMUX_DMAREQ_ID_UART8_TX	0x27	DMAMUX_DMAREQ_ID_SDIO1
0x67	DMAMUX_DMAREQ_ID_SDIO2	0x28	DMAMUX_DMAREQ_ID_QSPI1
0x68	DMAMUX_DMAREQ_ID_QSPI2	0x2A	DMAMUX_DMAREQ_ID_TMR1_CH1
0x2B	DMAMUX_DMAREQ_ID_TMR1_CH2	0x2C	DMAMUX_DMAREQ_ID_TMR1_CH3
0x2D	DMAMUX_DMAREQ_ID_TMR1_CH4	0x2E	DMAMUX_DMAREQ_ID_TMR1_OVERFLOW
0x2F	DMAMUX_DMAREQ_ID_TMR1_TRIG	0x30	DMAMUX_DMAREQ_ID_TMR1_HALL
0x31	DMAMUX_DMAREQ_ID_TMR8_CH1	0x32	DMAMUX_DMAREQ_ID_TMR8_CH2
0x33	DMAMUX_DMAREQ_ID_TMR8_CH3	0x34	DMAMUX_DMAREQ_ID_TMR8_CH4
0x35	DMAMUX_DMAREQ_ID_TMR8_OVERFLOW	0x36	DMAMUX_DMAREQ_ID_TMR8_TRIG
0x37	DMAMUX_DMAREQ_ID_TMR8_HALL	0x38	DMAMUX_DMAREQ_ID_TMR2_CH1
0x39	DMAMUX_DMAREQ_ID_TMR2_CH2	0x3A	DMAMUX_DMAREQ_ID_TMR2_CH3
0x3B	DMAMUX_DMAREQ_ID_TMR2_CH4	0x3C	DMAMUX_DMAREQ_ID_TMR2_OVERFLOW
0x7E	DMAMUX_DMAREQ_ID_TMR2_TRIG	0x3D	DMAMUX_DMAREQ_ID_TMR3_CH1
0x3E	DMAMUX_DMAREQ_ID_TMR3_CH2	0x3F	DMAMUX_DMAREQ_ID_TMR3_CH3
0x40	DMAMUX_DMAREQ_ID_TMR3_CH4	0x41	DMAMUX_DMAREQ_ID_TMR3_OVERFLOW
0x42	DMAMUX_DMAREQ_ID_TMR3_TRIG	0x43	DMAMUX_DMAREQ_ID_TMR4_CH1
0x44	DMAMUX_DMAREQ_ID_TMR4_CH2	0x45	DMAMUX_DMAREQ_ID_TMR4_CH3
0x46	DMAMUX_DMAREQ_ID_TMR4_CH4	0x47	DMAMUX_DMAREQ_ID_TMR4_OVERFLOW
0x7F	DMAMUX_DMAREQ_ID_TMR4_TRIG	0x48	DMAMUX_DMAREQ_ID_TMR5_CH1
0x49	DMAMUX_DMAREQ_ID_TMR5_CH2	0x4A	DMAMUX_DMAREQ_ID_TMR5_CH3

Request source ID	Description	Request source ID	Description
0x4B	DMAMUX_DMAREQ_ID_TMR5_CH4	0x4C	DMAMUX_DMAREQ_ID_TMR5_OVERFLOW
0x4D	DMAMUX_DMAREQ_ID_TMR5_TRIG	0x56	DMAMUX_DMAREQ_ID_TMR20_CH1
0x57	DMAMUX_DMAREQ_ID_TMR20_CH2	0x58	DMAMUX_DMAREQ_ID_TMR20_CH3
0x59	DMAMUX_DMAREQ_ID_TMR20_CH4	0x5A	DMAMUX_DMAREQ_ID_TMR20_OVERFLOW
0x5D	DMAMUX_DMAREQ_ID_TMR20_TRIG	0x5E	DMAMUX_DMAREQ_ID_TMR20_HALL
0x69	DMAMUX_DMAREQ_ID_DVP		

Example:

```
/* tmr20 hall dmamux function enable */
dma_flexible_config(DMA2, DMA1MUX_CHANNEL1, DMAMUX_DMAREQ_ID_TMR20_HALL);
```

5.8.11 dmamux_enable function

The table below describes the function dmamux_enable.

Table 187. dmamux_enable function

Name	Description
Function name	dmamux_enable
Function prototype	void dmamux_enable(dma_type *dma_x, confirm_state new_state);
Function description	Enable DMAMUX feature
Input parameter 1	dma_x: DMAx, x=1 or 2
Input parameter 2	new_state: enable or disable a channel
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

Enable or disable a DMA channel

FALSE: Channel disabled

TRUE: Channel enabled

Example:

```
/* dmamux function enable */
dmamux_enable(DMA2, TRUE);
```

5.8.12 dmamux_init function

The table below describes the function dmamux_init.

Table 188. dmamux_init function

Name	Description
Function name	dmamux_init
Function prototype	void dmamux_init(dmamux_channel_type *dmamux_channelx, dmamux_requst_id_sel_type dmamux_req_sel);
Function description	Configure DMAMUX
Input parameter 1	*dmamux_channelx* :DMAMUX channel selection, x=1…7
Input parameter 2	*dmamux_req_sel* :DMAMUX channel request ID
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* generator1 for dmamux channel4 as dma request   */
dmamux_init(DMA2MUX_CHANNEL4, DMAMUX_DMAREQ_ID_REQ_G1);
```

5.8.13 dmamux_sync_default_para_init function

The table below describes the function dmamux_sync_default_para_init.

Table 189. dmamux_sync_default_para_init function

Name	Description
Function name	dmamux_sync_default_para_init
Function prototype	void dmamux_sync_default_para_init(dmamux_sync_init_type *dmamux_sync_init_struct);
Function description	Initialize the parameters in the dmamux_sync_init_struct
Input parameter 1	dmamux_sync_init_struct: dmamux_sync_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below shows the default values of members in the dmamux_sync_init_struct.

Table 190. dmamux_sync_init_struct default values

Member	Default value
sync_enable	FALSE
sync_event_enable	FALSE
sync_polarity	DMAMUX_SYNC_POLARITY_DISABLE
sync_request_number	0x0
sync_signal_sel	(dmamux_sync_id_sel_type)0

Example:

```
/* dmamux sync init config with its default value */
dmamux_sync_init_type dmamux_sync_init_struct = {0};
dmamux_sync_default_para_init (&dmamux_sync_init_struct);
```

5.8.14 dmamux_sync_config function

The table below describes the function dmamux_sync_config.

Table 191. dmamux_sync_config function

Name	Description
Function name	dmamux_sync_config
Function prototype	void dmamux_sync_config(dmamux_channel_type *dmamux_channelx, dmamux_sync_init_type *dmamux_sync_init_struct);
Function description	Configure DMAMUX synchronous feature
Input parameter 1	*dmamux_channelx* :DMAMUX channel selection, x=1…7
Input parameter 2	dmamux_sync_init_struct: dmamux_sync_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dmamux_sync_init_struct

The dmamux_sync_init_type is defined in the at32f435_437_dma.h.

```
typedef struct
{
    dmamux_sync_id_sel_type      sync_signal_sel;
    uint32_t                     sync_polarity;
    uint32_t                     sync_request_number;
    confirm_state                sync_event_enable;
    confirm_state                sync_enable;
} dmamux_sync_init_type;
```

sync_signal_sel

Select signal source for synchronous module.

DMAMUX_SYNC_ID_EXINT0: External exint0 signal
DMAMUX_SYNC_ID_EXINT1: External exint1 signal
DMAMUX_SYNC_ID_EXINT2: External exint2 signal
DMAMUX_SYNC_ID_EXINT3: External exint3 signal
DMAMUX_SYNC_ID_EXINT4: External exint4 signal
DMAMUX_SYNC_ID_EXINT5: External exint5 signal
DMAMUX_SYNC_ID_EXINT6: External exint6 signal
DMAMUX_SYNC_ID_EXINT7: External exint7 signal
DMAMUX_SYNC_ID_EXINT8: External exint8 signal
DMAMUX_SYNC_ID_EXINT9: External exint9 signal
DMAMUX_SYNC_ID_EXINT10: External exint10 signal
DMAMUX_SYNC_ID_EXINT11: External exint11 signal
DMAMUX_SYNC_ID_EXINT12: External exint12 signal

DMAMUX_SYNC_ID_EXINT13: External exint13 signal
DMAMUX_SYNC_ID_EXINT14: External exint14 signal
DMAMUX_SYNC_ID_EXINT15: External exint15 signal
DMAMUX_SYNC_ID_DMAMUX_CH1_EVT: dmamux channel 1 event
DMAMUX_SYNC_ID_DMAMUX_CH2_EVT: dmamux channel 2 event
DMAMUX_SYNC_ID_DMAMUX_CH3_EVT: dmamux channel 3 event
DMAMUX_SYNC_ID_DMAMUX_CH4_EVT: dmamux channel 4 event
DMAMUX_SYNC_ID_DMAMUX_CH5_EVT: dmamux channel 5 event
DMAMUX_SYNC_ID_DMAMUX_CH6_EVT: dmamux channel 6 event
DMAMUX_SYNC_ID_DMAMUX_CH7_EVT: dmamux channel 7 event

sync_polarity

Polarity selection for synchronous signal
DMAMUX_SYNC_POLARITY_RISING: Rising edge
DMAMUX_SYNC_POLARITY_FALLING: Falling edge
DMAMUX_SYNC_POLARITY_RISING_FALLING: Rising edge and falling edge

sync_request_number

The number of DMA requests that can be synchronized
Range: 1~32

sync_event_enable

Enble or disable synchronous event generation
TURE: Synchronous event generated
FALSE: No synchronous event generated

sync_enable

Enable or disable synchronous module
FALSE: Disabled
TRUE: Enabled

Example:

```
dmamux_sync_default_para_init(&dmamux_sync_init_struct);
dmamux_sync_init_struct.sync_request_number = 4;
dmamux_sync_init_struct.sync_signal_sel = DMAMUX_SYNC_ID_EXINT1;
dmamux_sync_init_struct.sync_polarity = DMAMUX_SYNC_POLARITY_RISING;
dmamux_sync_init_struct.sync_event_enable = TRUE;
dmamux_sync_init_struct.sync_enable = TRUE;
dmamux_sync_config(DMA2MUX_CHANNEL4, &dmamux_sync_init_struct);
```

5.8.15 dmamux_generator_default_para_init function

The table below describes the function dmamux_generator_default_para_init.

Table 192. dmamux_generator_default_para_init function

Name	Description
Function name	dmamux_generator_default_para_init
Function prototype	void dmamux_generator_default_para_init(dmamux_gen_init_type *dmamux_gen_init_struct);
Function description	Initialize the parameters of the dmamux_gen_init_struct
Input parameter 1	dmamux_gen_init_struct: dmamux_gen_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The below table shows the default values of members in the dmamux_gen_init_struct.

Table 193. dmamux_gen_init_struct default values

Member	Default value
gen_signal_sel	(dmamux_gen_id_sel_type)0x0
gen_polarity	DMAMUX_GEN_POLARITY_DISABLE
gen_request_number	0x0
gen_enable	FALSE

Example:

```
/* dmamux gen init config with its default value */
dmamux_gen_init_type dmamux_gen_init_struct = {0};
dmamux_gen_default_para_init (&dmamux_gen_init_struct);
```

5.8.16 dmamux_generator_config function

The table below describes the function dmamux_generator_config.

Table 194. dmamux_generator_config function

Name	Description
Function name	dmamux_generator_config
Function prototype	void dmamux_generator_config(dmamux_generator_type * dmamux_gen_x, dmamux_gen_init_type *dmamux_gen_init_struct);
Function description	Configure DMAMUX request generator feature
Input parameter 1	*dmamux_gen_x* : request generator channel
Input parameter 2	dmamux_sync_init_struct: dmamux_sync_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dmamux_gen_x

dma request generator channel selection

DMA1MUX_GENERATOR1

DMA1MUX_GENERATOR2
DMA1MUX_GENERATOR3
DMA1MUX_GENERATOR4
DMA2MUX_GENERATOR1
DMA2MUX_GENERATOR2
DMA2MUX_GENERATOR3
DMA2MUX_GENERATOR4

dmamux_sync_init_struct

The dmamux_gen_init_type is defined in the at32f435_437_dma.h.

```
typedef struct
{
    dmamux_gen_id_sel_type          gen_signal_sel;
    uint32_t                        gen_polarity;
    uint32_t                        gen_request_number;
    confirm_state                   gen_enable;
} dmamux_gen_init_type;
```

gen_signal_sel

Select signal source for synchronous module

DMAMUX_GEN_ID_EXINT0: External exint0 singal
DMAMUX_GEN _ID_EXINT1: External exint0 singal
DMAMUX_GEN _ID_EXINT2: External exint2 singal
DMAMUX_GEN _ID_EXINT3: External exint3 singal
DMAMUX_GEN_ID_EXINT4: External exint4 singal
DMAMUX_GEN_ID_EXINT5: External exint5 singal
DMAMUX_GEN_ID_EXINT6: External exint6 singal
DMAMUX_GEN_ID_EXINT7: External exint7 singal
DMAMUX_GEN_ID_EXINT8: External exint8 singal
DMAMUX_GEN_ID_EXINT9: External exint9 singal
DMAMUX_GEN_ID_EXINT10: External exint10 singal
DMAMUX_GEN_ID_EXINT11: External exint11 singal
DMAMUX_GEN_ID_EXINT12: External exint12 singal
DMAMUX_GEN_ID_EXINT13: External exint13 singal
DMAMUX_GEN_ID_EXINT14: External exint14 singal
DMAMUX_GEN_ID_EXINT15: External exint15 singal
DMAMUX_GEN_ID_DMAMUX_CH1_EVT: dmamux channel 1 event
DMAMUX_GEN_ID_DMAMUX_CH2_EVT: dmamux channel 2 event
DMAMUX_GEN_ID_DMAMUX_CH3_EVT: dmamux channel 3 event
DMAMUX_GEN_ID_DMAMUX_CH4_EVT: dmamux channel 4 event
DMAMUX_GEN_ID_DMAMUX_CH5_EVT: dmamux channel 5 event
DMAMUX_GEN_ID_DMAMUX_CH6_EVT: dmamux channel 6 event
DMAMUX_GEN_ID_DMAMUX_CH7_EVT: dmamux channel 7 event

gen_polarity

Polarity selection for request generator signal

DMAMUX_GEN_POLARITY_RISING: Rising edge
DMAMUX_GEN_POLARITY_FALLING: Falling edge
DMAMUX_GEN_POLARITY_RISING_FALLING: Rising edge and falling edge

gen_request_number

The number of DMA requests that are generated by request generator

Range: 1~32

gen_enable

Enable or disable request generator

FALSE: Disabled

TRUE: Enabled

Example:

```
/* genertor1 configuration */
dmamux_generator_default_para_init(&dmamux_gen_init_struct);
dmamux_gen_init_struct.gen_polarity = DMAMUX_GEN_POLARITY_RISING;
dmamux_gen_init_struct.gen_request_number = 4;
dmamux_gen_init_struct.gen_signal_sel = DMAMUX_GEN_ID_EXINT1;
dmamux_gen_init_struct.gen_enable = TRUE;
dmamux_generator_config(DMA2MUX_GENERATOR1, &dmamux_gen_init_struct);
```

5.8.17 dmamux_sync_interrupt_enable function

The table below describes the function dmamux_sync_interrupt_enable.

Table 195. dmamux_sync_interrupt_enable function

Name	Description
Function name	dmamux_sync_interrupt_enable
Function prototype	void dmamux_sync_interrupt_enable(dmamux_channel_type *dmamux_channelx, confirm_state new_state);
Function description	Enable synchronous module overflow interrupt
Input parameter 1	*dmamux_channelx* :DMAMUX channel selection, x=1…7
Input parameter 2	new_state: enable or disable interrupts
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

Enable or disable DMA channel interrupts

FALSE: Interrupt disabled

TRUE: Interrupt enabled

Example:

```
/* enable sync overrun intterrupt */
dmamux_sync_interrupt_enable (DMA2MUX_CHANNEL1, TRUE);
```

5.8.18 dmamux_generator_interrupt_enable function

The table below describes the function dmamux_generator_interrupt_enable.

Table 196. dmamux_generator_interrupt_enable function

Name	Description
Function name	dmamux_generator_interrupt_enable
Function prototype	void dmamux_generator_interrupt_enable(dmamux_generator_type *dmamux_gen_x, confirm_state new_state);
Function description	Enable request generator overflow interrupt
Input parameter 1	*dmamux_gen_x* :DMAMUX request generator channel selection, x=1…4
Input parameter 2	new_state: enable or disable interrupts
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

Enable or disable request generator channel interrupts

FALSE: Interrupt disabled

TRUE: Interrupt enabled

Example:

```
/* enable gen overrun intterrupt */
dmamux_generator_interrupt_enable(DMA2MUX_GENERATOR3, TRUE);
```

5.8.19 dmamux_sync_flag_get function

The table below describes the function dmamux_sync_flag_get.

Table 197. dmamux_sync_flag_get function

Name	Description
Function name	dmamux_sync_flag_get
Function prototype	flag_status dmamux_sync_flag_get(dma_type *dma_x, uint32_t flag);
Function description	Get dmamux synchronous flag
Input parameter 1	dma_x: DMAx, x=1 or 2
Input parameter 2	flag: flag selection
Output parameter	NA
Return value	flag_status: indicates whether or not the selected flag is set
Required preconditions	NA
Called functions	NA

flag

This is used for flag selection, including:

DMAMUX_SYNC_OV1_FLAG

DMAMUX_SYNC_OV2_FLAG

DMAMUX_SYNC_OV3_FLAG

DMAMUX_SYNC_OV4_FLAG

DMAMUX_SYNC_OV5_FLAG

DMAMUX_SYNC_OV6_FLAG
DMAMUX_SYNC_OV7_FLAG

flag_status

RESET: Corresponding flag is not set
SET: Corresponding flag is set

Example:

```
if(dmamux_sync_flag_get (DMA2, DMAMUX_SYNC_OV1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
}
```

5.8.20 dmamux_sync_flag_clear function

The table below describes the function dmamux_sync_flag_clear.

Table 198. dmamux_sync_flag_clear function

Name	Description
Function name	dmamux_sync_flag_clear
Function prototype	void dmamux_sync_flag_clear(dma_type *dma_x, uint32_t flag);
Function description	Clear synchronous module flag
Input parameter 1	dma_x: DMAx, x=1 or 2
Input parameter 2	flag: flag selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flag

This is used for flag selection, including:
DMAMUX_SYNC_OV1_FLAG
DMAMUX_SYNC_OV2_FLAG
DMAMUX_SYNC_OV3_FLAG
DMAMUX_SYNC_OV4_FLAG
DMAMUX_SYNC_OV5_FLAG
DMAMUX_SYNC_OV6_FLAG
DMAMUX_SYNC_OV7_FLAG

Example:

```
if(dmamux_sync_flag_get (DMA2, DMAMUX_SYNC_OV1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
    dmamux_sync_flag_clear(DMA2, DMAMUX_SYNC_OV1_FLAG);
}
```

5.8.21 dmamux_generator_flag_get function

The table below describes the function dmamux_generator_flag_get.

Table 199. dmamux_generator_flag_get function

Name	Description
Function name	dmamux_ generator_flag_get
Function prototype	flag_status dmamux_ generator_flag_get(dma_type *dma_x, uint32_t flag);
Function description	Get dmamux request generator flag
Input parameter 1	dma_x: DMAx, x=1 or 2
Input parameter 2	flag: flag selection
Output parameter	NA
Return value	flag_status: indicates whether or not the selected flag is set
Required preconditions	NA
Called functions	NA

flag

This is used for flag selection, including:

DMAMUX_GEN_TRIG_OV1_FLAG

DMAMUX_GEN_TRIG_OV2_FLAG

DMAMUX_GEN_TRIG_OV3_FLAG

DMAMUX_GEN_TRIG_OV4_FLAG

flag_status

RESET: Corresponding flag is not set

SET: Corresponding flag is set

Example:

```
if(dmamux_generator_flag_get (DMA2, DMAMUX_GEN_TRIG_OV1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
}
```

5.8.22 dmamux_generator_flag_clear function

The table below describes the function dmamux_generator_flag_clear.

Table 200. dmamux_generator_flag_clear function

Name	Description
Function name	dmamux_generator_flag_clear
Function prototype	void dmamux_generator_flag_clear(dma_type *dma_x, uint32_t flag);
Function description	Clear request generator flag
Input parameter 1	dma_x: DMAx, x=1 or 2
Input parameter 2	flag: flag selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flag

This is used for flag selection, including:

DMAMUX_GEN_TRIG_OV1_FLAG

DMAMUX_GEN_TRIG_OV2_FLAG

DMAMUX_GEN_TRIG_OV3_FLAG

DMAMUX_GEN_TRIG_OV4_FLAG

Example:

```
if(dmamux_generator_flag_get (DMA2, DMAMUX_GEN_TRIG_OV1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
    dmamux_generator_flag_clear(DMA2, DMAMUX_GEN_TRIG_OV1_FLAG);
}
```

5.9 Digital video parallel interface (DVP)

The DVP register structure dvp_type is defined in the “at32f435_437_dvp.h”.

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


} dvp_type;
```

The table below gives a list of the DVP registers.

Table 201. Summary of DVP registers

Register	Description
ctrl	DVP control register
sts	DVP status register
ests	DVP event status register
iena	DVP interrupt enable register
ists	DVP interrupt status register
iclr	DVP interrupt clear register
scr	DVP embedded synchronization code register
sur	DVP mbedded synchronization unmask register
cwst	DVP crop window start register
cwsz	DVP crop window size register
dt	DVP data register
actrl	DVP advanced control register
hscf	DVP enhanced horizontal scaling factor register
vscf	DVP enhanced vertical scaling factor register
frf	DVP enhanced frame rate control factor register
bth	DVP binarization threshold register

The table below gives a list of DVP library functions.

Table 202. Summary of CVP library functions

Function name	Description
dvp_reset	Reset DVP by CRM reset register
dvp_capture_enable	Enable/disable DVP capture
dvp_capture_mode_set	Configure DVP capture mode
dvp_window_crop_enable	Enable/disable DVP crop window
dvp_window_crop_set	Configure DVP crop window
dvp_jpeg_enable	Enable/disable DVP JPEG mode
dvp_sync_mode_set	Configure DVP synchronization mode
dvp_sync_code_set	Configure DVP embedded synchronization code
dvp_sync_unmask_set	Configure DVP synchronization code unmask
dvp_pclk_polarity_set	Configure DVP pixel clock polarity

dvp_hsync_polarity_set	Configure DVP horizontal synchronization polarity
dvp_vsync_polarity_set	Configure DVP vertical synchronization polarity
dvp_basic_frame_rate_control_set	Configure DVP basic frame rate control
dvp_pixel_data_length_set	Configure DVP pixel data length
dvp_enable	Enable/disable DVP
dvp_zoomout_select	Configure the basic pixel capture/drop extended selection
dvp_zoomout_set	Configure DVP capture/drop feature
dvp_basic_status_get	Get the DVP basic status
dvp_interrupt_enable	Enable/disable DVP interrupts
dvp_interrupt_flag_get	Get DVP event interrupt flag
dvp_flag_get	Get DVP event/interrupt flag
dvp_flag_clear	Clear DVP event/interrupt flag
dvp_enhanced_scaling_resize_enable	Enable/disable DVP enhanced image reduction feature
dvp_enhanced_scaling_resize_set	Configure the DVP enhanced image reduction size
dvp_enhanced_framerate_set	Configure enhanced frame rate control feature
dvp_monochrome_image_binarization_set	Configure DVP monochrome image binarization threshold
dvp_enhanced_data_format_set	Configure DVP enhanced function data format
dvp_input_data_unused_set	Configure DVP input data unused setting and bit number
dvp_dma_burst_set	Configure DVP burst DMA transfer
dvp_sync_event_interrupt_set	Configure DVP HSYNC/VSYNC event and interrupt definition

5.9.1 dvp_reset function

The table below describes the function dvp_reset.

Table 203. dvp_reset function

Name	Description
Function name	dvp_reset
Function prototype	void dvp_reset(void);
Function description	Reset DVP by CRM reset register
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

Example:

```
dvp_reset();
```

5.9.2 dvp_capture_enable function

The table below describes the function dvp_capture_enable.

Table 204. dvp_capture_enable function

Name	Description
Function name	dvp_capture_enable
Function prototype	void dvp_capture_enable(confirm_state new_state);
Function description	Enable/disable DVP capture
Input parameter	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dvp_capture_enable(TRUE);
```

5.9.3 dvp_capture_mode_set function

The table below describes the function dvp_capture_mode_set.

Table 205. dvp_capture_mode_set function

Name	Description
Function name	dvp_capture_mode_set
Function prototype	void dvp_capture_mode_set(dvp_cfm_type cap_mode);
Function description	Configure DVP capture mode
Input parameter	cap_mode: DVP capture mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

cap_mode

This is used to configure the DVP capture mode.

DVP_CAP_FUNC_MODE_CONTINUOUS: Continuous capture mode

DVP_CAP_FUNC_MODE_SINGLE: Single frame capture mode

Example:

```
dvp_capture_mode_set(DVP_CAP_FUNC_MODE_CONTINUOUS);
```

5.9.4 dvp_window_crop_enable function

The table below describes the function dvp_window_crop_enable.

Table 206. dvp_window_crop_enable function

Name	Description
Function name	dvp_window_crop_enable
Function prototype	void dvp_window_crop_enable(confirm_state new_state);
Function description	Enable/disable DVP crop window
Input parameter	new_state: Enable (TRUE) or disable (FALSE)

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

Example:

```
dvp_window_crop_enable(TRUE);
```

5.9.5 dvp_window_crop_set function

The table below describes the function dvp_window_crop_set.

Table 207. dvp_window_crop_set function

Name	Description
Function name	dvp_window_crop_set
Function prototype	void dvp_window_crop_set(uint16_t crop_x, uint16_t crop_y, uint16_t crop_w, uint16_t crop_h, uint8_t bytes);
Function description	Configure DVP crop window
Input parameter 1	crop_x: Crop window horizontal start pixel, range: 0x0000~0x3FFF/bytes
Input parameter 2	crop_y: Crop window vertical start pixel, range: 0x0000~0x1FFF
Input parameter 3	crop_w: Crop window horizontal pixel number minus one, range: 0x0001~(0x40000)/bytes
Input parameter 4	crop_h: Crop window vertical line number minus one, range: 0x0001~0x40000
Input parameter 5	bytes: number of bytes of one pixel, for example, in Y8 format, one pixel corresponds to 1 byte, therefore bytes = 1; in RGB565 format, one pixel corresponds to 2 bytes, therefore bytes = 2.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dvp_window_crop_set(0x0F, 0x0F, 0x100, 0x100, 0x02);
```

5.9.6 dvp_jpeg_enable function

The table below describes the function dvp_jpeg_enable.

Table 208. dvp_jpeg_enable function

Name	Description
Function name	dvp_jpeg_enable
Function prototype	void dvp_jpeg_enable(confirm_state new_state);
Function description	Enable/disable DVP JPEG mode
Input parameter	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dvp_jpeg_enable(TRUE);
```

5.9.7 dvp_sync_mode_set function

The table below describes the function dvp_sync_mode_set.

Table 209. dvp_sync_mode_set function

Name	Description
Function name	dvp_sync_mode_set
Function prototype	void dvp_sync_mode_set(dvp_sm_type sync_mode);
Function description	Configure DVP synchronization mode
Input parameter	sync_mode: DVP synchronization mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

sync_mode

This is used to select the DVP synchronization mode.

DVP_SYNC_MODE_HARDWARE: Hardware synchronization

DVP_SYNC_MODE_EMBEDDED: Embedded code synchronization

Example:

```
dvp_sync_mode_set(DVP_SYNC_MODE_HARDWARE);
```

5.9.8 dvp_sync_code_set function

The table below describes the function dvp_sync_code_set.

Table 210. dvp_sync_code_set function

Name	Description
Function name	dvp_sync_code_set
Function prototype	void dvp_sync_code_set(uint8_t fmsc, uint8_t fmec, uint8_t lnsc, uint8_t lnec);
Function description	Configure DVP embedded code synchronization
Input parameter 1	fmsc: Frame start synchronization code, range: 0x00~0xFF
Input parameter 2	fmec: Frame end synchronization code, range: 0x00~0xFF
Input parameter 3	lnsc: Line start synchronization code, range: 0x00~0xFF
Input parameter 4	lnec: Line end synchronization code, range: 0x00~0xFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dvp_sync_code_set(0xFF, 0xFF, 0xC7, 0xDA);
```

5.9.9 dvp_sync_unmask_set function

The table below describes the function dvp_sync_unmask_set.

Table 211. dvp_sync_unmask_set function

Name	Description
Function name	dvp_sync_unmask_set
Function prototype	void dvp_sync_unmask_set(uint8_t fmsu, uint8_t fmeu, uint8_t lnsu, uint8_t lneu);
Function description	Configure DVP embedded synchronization code unmask
Input parameter 1	fmsu: Frame start synchronization code unmask, range: 0x00~0xFF
Input parameter 2	fmeu: Frame end synchronization code unmask, range: 0x00~0xFF
Input parameter 3	lnsu: Line start synchronization code unmask, range: 0x00~0xFF
Input parameter 4	lneu: Line end synchronization code unmask, range: 0x00~0xFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dvp_sync_unmask_set(0x0F, 0x0F, 0xF0, 0xF0);
```

5.9.10 dvp_pclk_polarity_set function

The table below describes the function dvp_pclk_polarity_set.

Table 212. dvp_pclk_polarity_set function

Name	Description
Function name	dvp_pclk_polarity_set
Function prototype	void dvp_pclk_polarity_set(dvp_ckp_type eage);
Function description	Configure DVP pixel clock polarity
Input parameter	eage: DVP pixel clock polarity
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

eage

This is used to select the DVP pixel clock polarity.

DVP_CLK_POLARITY_RISING: Input data capture on DVP_PCLK rising edge

DVP_CLK_POLARITY_FALLING: Input data capture on DVP_PCLK falling edge

Example:

```
dvp_pclk_polarity_set(DVP_CLK_POLARITY_RISING);
```

5.9.11 dvp_hsync_polarity_set function

The table below describes the function dvp_hsync_polarity_set.

Table 213. dvp_hsync_polarity_set function

Name	Description
Function name	dvp_hsync_polarity_set
Function prototype	void dvp_hsync_polarity_set(dvp_hsp_type hsync_pol);
Function description	Configure DVP horizontal synchronization polarity
Input parameter	hsync_pol: DVP horizontal synchronization polarity
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

hsync_pol

This is used to select the DVP horizontal synchronization polarity.

DVP_HSYNC_POLARITY_HIGH: DVP_HSYNC high level indicates that the captured data is a valid pixel data, and Line start signal on the rising edge

DVP_HSYNC_POLARITY_LOW: DVP_HSYNC low level indicates that the captured data is a valid pixel data, and Line start signal on the falling edge

Example:

```
dvp_hsync_polarity_set(DVP_HSYNC_POLARITY_HIGH);
```

5.9.12 dvp_vsync_polarity_set function

The table below describes the function dvp_vsync_polarity_set.

Table 214. dvp_vsync_polarity_set function

Name	Description
Function name	dvp_vsync_polarity_set
Function prototype	void dvp_vsync_polarity_set(dvp_vsp_type vsync_pol);
Function description	Configure DVP vertical synchronization polarity
Input parameter	vsync_pol: DVP vertical synchronization polarity
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

vsync_pol

This is used to select DVP vertical synchronization polarity.

Frame start:

DVP_VSYNC_POLARITY_LOW: DVP_VSYNC low level indicates a Frame start signal

DVP_VSYNC_POLARITY_HIGH: DVP_VSYNC high level indicates a Frame start signal

Frame effective:

DVP_VSYNC_POLARITY_LOW: DVP_VSYNC low level indicates that the captured data is in vertical blanking

DVP_VSYNC_POLARITY_HIGH: DVP_VSYNC high level indicates that the captured data is in vertical blanking

Example:

```
dvp_vsync_polarity_set(DVP_VSYNC_POLARITY_LOW);
```

5.9.13 dvp_basic_frame_rate_control_set function

The table below describes the function dvp_basic_frame_rate_control_set.

Table 215. dvp_basic_frame_rate_control_set function

Name	Description
Function name	dvp_basic_frame_rate_control_set
Function prototype	void dvp_basic_frame_rate_control_set(dvp_bfrc_type dvp_bfrc);
Function description	Configure DVP basic frame rate control
Input parameter	dvp_bfrc: DVP basic frame rate control
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dvp_bfrc

This is used to select DVP basic frame rate control.

DVP_BFRC_ALL: All frames are captured or use enhanced frame rate control feature

DVP_BFRC_HALF: Enable frame rate control, every alternate frame captured

DVP_BFRC_QUARTER: Enable frame rate control, one frame in 4 frames captured

Example:

```
dvp_basic_frame_rate_control_set(DVP_BFRC_ALL);
```

5.9.14 dvp_pixel_data_length_set function

The table below describes the function dvp_pixel_data_length_set.

Table 216. dvp_pixel_data_length_set function

Name	Description
Function name	dvp_pixel_data_length_set
Function prototype	void dvp_pixel_data_length_set(dvp_pdl_type dvp_pdl);
Function description	Configure DVP pixel data length
Input parameter	dvp_pdl: DVP pixel data length
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dvp_pdl

This is used to select DVP pixel data length.

DVP_PIXEL_DATA_LENGTH_8: Interface captures 8-bit data

DVP_PIXEL_DATA_LENGTH_10: Interface captures 10-bit data

DVP_PIXEL_DATA_LENGTH_12: Interface captures 12-bit data

DVP_PIXEL_DATA_LENGTH_14: Interface captures 14-bit data

Example:

```
dvp_pixel_data_length_set(DVP_PIXEL_DATA_LENGTH_8);
```

5.9.15 dvp_enable function

The table below describes the function dvp_enable.

Table 217. dvp_enable function

Name	Description
Function name	dvp_enable
Function prototype	void dvp_enable(confirm_state new_state);
Function description	Enable/disable DVP
Input parameter	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dvp_enable(TRUE);
```

5.9.16 dvp_zoomout_select function

The table below describes the function dvp_zoomout_select.

Table 218. dvp_zoomout_select function

Name	Description
Function name	dvp_zoomout_select
Function prototype	void dvp_zoomout_select(dvp_pcdes_type dvp_pcdes);
Function description	Configure the basic pixel capture/drop extended selection
Input parameter	dvp_pcdes: extended DVP pixel capture/drop configuration
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dvp_pcdes

This is used for the basic pixel capture/drop extended selection.

PCDS = 0:

DVP_PCDES_CAP_FIRST: Capture the first pixel data and drop others

DVP_PCDES_DROP_FIRST: Capture the second pixel data and drop others

PCDS = 1:

DVP_PCDES_CAP_FIRST: Capture the third pixel data and drop others

DVP_PCDES_DROP_FIRST: Capture the fourth pixel data and drop others

Example:

```
dvp_zoomout_select(DVP_PCDES_CAP_FIRST);
```

5.9.17 dvp_zoomout_set function

The table below describes the function dvp_zoomout_set.

Table 219. dvp_zoomout_set function

Name	Description
Function name	dvp_zoomout_set
Function prototype	void dvp_zoomout_set(dvp_pcdc_type dvp_pcdc, dvp_pcds_type dvp_pcds, dvp_lcdc_type dvp_lcdc, dvp_lcds_type dvp_lcds);
Function description	Configure DVP capture/drop feature
Input parameter 1	dvp_pcdc: basic pixel capture/drop control
Input parameter 2	dvp_pcds: basic pixel capture/drop selection
Input parameter 3	dvp_lcdc: basic line capture/drop control
Input parameter 4	dvp_lcds: basic line capture/drop selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dvp_pcdc

This is used to select basic pixel capture/drop control.

DVP_PCDC_ALL: All frames are captured or use enhanced image scaling resize feature

DVP_PCDC_ONE_IN_TWO: Enable capture/drop control to capture one in two pixel data

DVP_PCDC_ONE_IN_FOUR: Enable capture/drop control to capture one in four pixel data

DVP_PCDC_TWO_IN_FOUR:

Enable capture/drop control to capture two consecutive data in four pixel data

dvp_pcds

This is used for basic pixel capture/drop selection.

DVP_PCDS_CAP_FIRST:

Capture the first group of data (one or two pixel data) and drop the next group

DVP_PCDS_DROP_FIRST:

Drop the first group of data (one or two pixel data) and capture the next group

dvp_lcdc

This is used for basic line capture/drop control.

DVP_LCDC_ALL:

All frames are captured or use enhanced image scaling resize feature

DVP_LCDC_ONE_IN_TWO:

Enable capture/drop control to capture one in two lines

dvp_lcds

This is used for basic line capture/drop selection.

DVP_LCDS_CAP_FIRST:

Capture the first line and drop the next line

DVP_LCDS_DROP_FIRST:

Drop the first line and capture the first line

Example:

```
dvp_zoomout_set(DVP_PCDC_ONE_IN_TWO, DVP_PCDS_CAP_FIRST, DVP_LCDC_ONE_IN_TWO,
DVP_LCDS_CAP_FIRST);
```

5.9.18 dvp_basic_status_get function

The table below describes the function dvp_basic_status_get.

Table 220. dvp_basic_status_get function

Name	Description
Function name	dvp_basic_status_get
Function prototype	flag_status dvp_basic_status_get(dvp_status_basic_type dvp_status_basic);
Function description	Get the DVP basic status
Input parameter	dvp_status_basic: DVP basic status
Output parameter	NA
Return value	Return SET or RESET.
Required preconditions	NA
Called functions	NA

dvp_status_basic

This is used to select the DVP basic status.

DVP_STATUS_HSYN: Horizontal synchonization status

DVP_STATUS_VSYN: Vertical synchonization status

DVP_STATUS_OFNE: Output data FIFO statu

Example:

```
flag_status dvp_status;
dvp_status = dvp_basic_status_get(DVP_STATUS_HSYN);
```

5.9.19 dvp_interrupt_enable function

The table below describes the function dvp_interrupt_enable.

Table 221. dvp_interrupt_enable function

Name	Description
Function name	dvp_interrupt_enable
Function prototype	void dvp_interrupt_enable(uint32_t dvp_int, confirm_state new_state);
Function description	Enable/disable DVP interrupt
Input parameter 1	dvp_int: DVP interrupt
Input parameter 2	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dvp_int

This is used to select the DVP interrupt.

DVP_CFD_INT: DVP capture frame done interrupt

DVP_OVR_INT: DVP output data FIFO overrun interrupt

DVP_ESE_INT: DVP embedded synchronization error interrupt

DVP_VS_INT: DVP Vertical synchonization interrupt

DVP_HS_INT: DVP horizontal synchonization interrupt

Example:

```
dvp_interrupt_enable(DVP_CFD_INT, TRUE);
```

5.9.20 dvp_interrupt_flag_get function

The table below describes the function dvp_interrupt_flag_get.

Table 222. dvp_interrupt_flag_get function

Name	Description
Function name	dvp_interrupt_flag_get
Function prototype	flag_status dvp_interrupt_flag_get(uint32_t flag);
Function description	Get DVP interrupt flag
Input parameter	flag: DVP event/interrupt flag
Output parameter	NA
Return value	Return SET or RESET.
Required preconditions	NA
Called functions	NA

flag

This is used to select the DVP event/interrupt flag.

DVP_CFD_INT_FLAG: Capture frame done interrupt status

DVP_OVR_INT_FLAG: Output data FIFO overrun interrupt status

DVP_ESE_INT_FLAG: Embedded synchronization error interrupt status

DVP_VS_INT_FLAG: Vertical synchonization interrupt status

DVP_HS_INT_FLAG: Horizontal synchonization interrupt status

Example:

```
flag_status dvp_flag;
dvp_flag = dvp_interrupt_flag_get(DVP_CFD_INT_FLAG);
```

5.9.21 dvp_flag_get function

The table below describes the function dvp_flag_get.

Table 223. dvp_flag_get function

Name	Description
Function name	dvp_flag_get
Function prototype	flag_status dvp_flag_get(uint32_t flag);
Function description	Get the DVP event/interrupt flag
Input parameter	flag: DVP event/interrupt flag
Output parameter	NA
Return value	Return SET or RESET.
Required preconditions	NA
Called functions	NA

flag

This is used to select the DVP event/interrupt flag.

DVP_CFD_EVT_FLAG: Capture frame done raw event status

DVP_OVR_EVT_FLAG: Output data FIFO overrun event status

DVP_ESE_EVT_FLAG: Embedded synchronization error event status

DVP_VS_EVT_FLAG: Vertical synchonization event status

DVP_HS_EVT_FLAG: Horizontal synchonization event status

Example:

```
flag_status dvp_flag;
```

```
dvp_flag = dvp_flag_get(DVP_CFD_EVT_FLAG);
```

5.9.22 dvp_flag_clear function

The table below describes the function dvp_flag_clear.

Table 224. dvp_flag_clear function function

Name	Description
Function name	dvp_flag_clear
Function prototype	void dvp_flag_clear(uint32_t flag);
Function description	Clear DVP event/ interrupt flag
Input parameter	flag: DVP event/ interrupt flag to be cleared; refer to *flag*
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dvp_flag_clear(DVP_CFD_EVT_FLAG);
```

5.9.23 dvp_enhanced_scaling_resize_enable function

The table below describes the function dvp_enhanced_scaling_resize_enable.

Table 225. dvp_enhanced_scaling_resize_enable function

Name	Description
Function name	dvp_enhanced_scaling_resize_enable
Function prototype	void dvp_enhanced_scaling_resize_enable(confirm_state new_state);
Function description	Enable/disable DVP enhanced scaling resize feature
Input parameter	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dvp_enhanced_scaling_resize_enable(TRUE);
```

5.9.24 dvp_enhanced_scaling_resize_set function

The table below describes the function dvp_enhanced_scaling_resize_set.

Table 226. dvp_enhanced_scaling_resize_set function

Name	Description
Function name	dvp_enhanced_scaling_resize_set
Function prototype	void dvp_enhanced_scaling_resize_set(uint16_t src_w, uint16_t des_w, uint16_t src_h, uint16_t des_h);
Function description	Configure the DVP enhanced scaling resize
Input parameter 1	src_w: Horizontal scaling resize source factor, range:0x0001~0x1FFF
Input parameter 2	des_w: Horizontal scaling resize target factor, range: 0x0001~0x1FFF
Input parameter 3	src_h: Vertical scaling resize source factor, range: 0x0001~0x1FFF
Input parameter 4	des_h: Vertical scaling resize target factor, range: 0x0001~0x1FFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dvp_enhanced_scaling_resize_set(0xFFF, 0x0F, 0xFFF, 0x0F);
```

5.9.25 dvp_enhanced_framerate_set function

The table below describes the function dvp_enhanced_framerate_set.

Table 227. dvp_enhanced_framerate_set function

Name	Description
Function name	dvp_enhanced_framerate_set
Function prototype	void dvp_enhanced_framerate_set(uint16_t efrcsf, uint16_t efrctf, confirm_state new_state);
Function description	Configure the enhanced frame rate control
Input parameter 1	efrcsf: Enhanced frame rate contorl source factor, range: 0x00~0x1F
Input parameter 2	efrctf: Enhanced frame rate control target factor, range: 0x00~0x1F
Input parameter 3	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dvp_enhanced_framerate_set(0x0F, 0x0F , TRUE);
```

5.9.26 dvp_monochrome_image_binarization_set function

The table below describes the function dvp_monochrome_image_binarization_set.

Table 228. dvp_monochrome_image_binarization_set function

Name	Description
Function name	dvp_monochrome_image_binarization_set
Function prototype	void dvp_monochrome_image_binarization_set(uint8_t mibthd, confirm_state new_state);
Function description	Configure the DVP monochrome image binarization threshold
Input parameter 1	mibthd: DVP monochrome image binarization threshold, range: 0x00~0xFF
Input parameter 2	new_state: DVP monochrome image binarization threshold status; enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
dvp_monochrome_image_binarization_set(0x0F, TRUE);
```

5.9.27 dvp_enhanced_data_format_set function

The table below describes the function dvp_enhanced_data_format_set.

Table 229. dvp_enhanced_data_format_set function

Name	Description
Function name	dvp_enhanced_data_format_set
Function prototype	void dvp_enhanced_data_format_set(dvp_efdf_type dvp_efdf);
Function description	Configure the DVP enhanced function data format
Input parameter	dvp_efdf: DVP enhanced function data format
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dvp_efdf

This is used to select the DVP enhanced function data format.

DVP_EFDF_BYPASS: Enhanced function data management disabled
DVP_EFDF_YUV422_UYVY: YUV422 (UYVY / VYUY) format data
DVP_EFDF_YUV422_YUYV: YUV422 (YUYV / YVYU) format data
DVP_EFDF_RGB565_555: RGB565 RGB555 format data
DVP_EFDF_Y8: Y8 (Y only) format data

Example:

```
dvp_enhanced_data_format_set(DVP_EFDF_RGB565_555);
```

5.9.28 dvp_input_data_unused_set function

The table below describes the function dvp_input_data_unused_set.

Table 230. dvp_input_data_unused_set function

Name	Description
Function name	dvp_input_data_unused_set
Function prototype	void dvp_input_data_unused_set(dvp_idus_type dvp_idus, dvp_idun_type dvp_idun);
Function description	Configure DVP input data unused setting and bit number
Input parameter 1	dvp_idus: input data unused setting
Input parameter 2	dvp_idun: input data unused bit number
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dvp_idus

This is used to select the input data unused setting.

DVP_IDUS_MSB: Unused data bit in the MSB
DVP_IDUS_LSB: Unused data bit in the LSB

dvp_idun

This is used to select the input data unused bit number.

DVP_IDUN_0: No unused bit
DVP_IDUN_2: 2-bit unused data

DVP_IDUN_4: 4-bit unused data

DVP_IDUN_6: 6-bit unused data

Example:

```
dvp_input_data_unused_set(DVP_IDUS_MSB, DVP_IDUN_2);
```

5.9.29 dvp_dma_burst_set function

The table below describes the function dvp_dma_burst_set.

Table 231. dvp_dma_burst_set function

Name	Description
Function name	dvp_dma_burst_set
Function prototype	void dvp_dma_burst_set(dvp_dmabt_type dvp_dmabt);
Function description	Configure DVP burst DMA transfer
Input parameter	dvp_dmabt: DVP burst DMA transfer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dvp_dmabt

This is used for DVP burst DMA transfer configuration.

DVP_DMABT_SINGLE: Burst DMA transfer configuration disabled

DVP_DMABT_BURST: Burst DMA transfer configuration enabled

Example:

```
dvp_dma_burst_set(DVP_DMABT_BURST);
```

5.9.30 dvp_sync_event_interrupt_set function

The table below describes the function dvp_sync_event_interrupt_set.

Table 232. dvp_sync_event_interrupt_set function

Name	Description
Function name	dvp_sync_event_interrupt_set
Function prototype	void dvp_sync_event_interrupt_set(dvp_hseid_type dvp_hseid, dvp_vseid_type dvp_vseid);
Function description	Configure the DVP HSYNC/VSYNC event and interrupt definition
Input parameter 1	dvp_hseid: Horizontal synchonization event and interrupt definition
Input parameter 2	dvp_vseid: Vertical synchonization event and interrupt definition
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dvp_hseid

This is used to select horizontal synchonization event and interrupt definition.

DVP_HSEID_LINE_END: HSES and HEIS indicate line end event and interrupt

DVP_HSEID_LINE_START: HSES and HEIS indicate line start event and interrupt

dvp_vseid

This is used to select vertical synchonization event and interrupt definition.

DVP_VSEID_FRAME_END: VSES and VEIS indicate frame end event and interrupt

DVP_VSEID_FRMAE_START: VSES and VEIS indicate frame start event and interrupt

Example:

```
dvp_sync_event_interrupt_set(DVP_HSEID_LINE_START, DVP_VSEID_FRAME_END);
```

5.10 EDMA controller (EDMA)

The EDMA register structure edma_type is defined in the “at32f435_437_edma.h”.

```
/**
  * @brief type define edma register all
  */
typedef struct
{
……
} edma_type;
```

The EDMA stream register structure edma_stream_type is defined in the “at32f435_437_edma.h”.

```
/**
  * @brief type define edma stream register all
  */
typedef struct
{
……
} edma_stream_type;
```

The EDMA linked table control register structure edma_stream_link_list_type is defined in the “at32f435_437_edma.h”.

```
/**
  * @brief type define edma stream link list pointer register
  */
typedef struct
{
……
} edma_stream_link_list_type;
```

The EDMA 2D transfer control register structure edma_stream_2d_type is defined in the “at32f435_437_edma.h”.

```
/**
  * @brief type define edma 2d register all
  */
typedef struct
{
……
} edma_stream_2d_type;
```

The EDMAMUX channel control register structure edmamux_channel_type is defined in the “at32f435_437_edma.h”.

```
/**
  * @brief type define edmamux muxsctrl register
  */
typedef struct
{
```

……
} edmamux_channel_type;

The EDMAMUX generator control register structure edmamux_generator_type is defined in the “at32f435_437_edma.h”.

```
/**
  * @brief type define edmamux request generator register all
  */
typedef struct
{
……
} edmamux_generator_type;
```

The table below gives a list of the EDMA registers.

Table 233. Summary of EDMA registers

Register	Description
edma_sts1	EDMA status register 1
edma_sts2	EDMA status register 2
edma_clr1	EDMA status clear register 1
edma_clr2	EDMA status clear register 2
edma_s1ctrl	EDMA stream 1 control register
edma_s1dtcnt	EDMA stream 1 number of data register
edma_s1paddr	EDMA stream 1 peripheral address register
edma_s1m0addr	EDMA stream 1 memory 0 address register
edma_s1m1addr	EDMA stream 1 memory 1 address register
edma_s1fctrl	EDMA stream 1 FIFO control register
edma_s2ctrl	EDMA stream 2 control register
edma_s2dtcnt	EDMA stream 2 number of data register
edma_s2paddr	EDMA stream 2 peripheral address register
edma_s2m0addr	EDMA stream 2 memory 0 address register
edma_s2m1addr	EDMA stream 2 memory 1 address register
edma_s2fctrl	EDMA stream 2 FIFO control register
edma_s3ctrl	EDMA stream 3 control register
edma_s3dtcnt	EDMA stream 3 number of data register
edma_s3paddr	EDMA stream 3 peripheral address register
edma_s3m0addr	EDMA stream 3 memory 0 address register
edma_s3m1addr	EDMA stream 3 memory 1 address register
edma_s3fctrl	EDMA stream 3 FIFO control register
edma_s4ctrl	EDMA stream 4 control register
edma_s4dtcnt	EDMA stream 4 number of data register
edma_s4paddr	EDMA stream 4 peripheral address register
edma_s4m0addr	EDMA stream 4 memory 0 address register
edma_s4m1addr	EDMA stream 4 memory 1 address register
edma_s4fctrl	EDMA stream 4 FIFO control register
edma_s5ctrl	EDMA stream 5 control register
edma_s5dtcnt	EDMA stream 5 number of data register

Register	Description
edma_s5paddr	EDMA stream 5 peripheral address register
edma_s5m0addr	EDMA stream 5 memory 0 address register
edma_s5m1addr	EDMA stream 5 memory 1 address register
edma_s5fctrl	EDMA stream 5 FIFO control register
edma_s6ctrl	EDMA stream 6 control register
edma_s6dtcnt	EDMA stream 6 number of data register
edma_s6paddr	EDMA stream 6 peripheral address register
edma_s6m0addr	EDMA stream 6 memory 0 address register
edma_s6m1addr	EDMA stream 6 memory 1 address register
edma_s6fctrl	EDMA stream 6 FIFO control register
edma_s7ctrl	EDMA stream 7 control register
edma_s7dtcnt	EDMA stream 7 number of data register
edma_s7paddr	EDMA stream 7 peripheral address register
edma_s7m0addr	EDMA stream 7 memory 0 address register
edma_s7m1addr	EDMA stream 7 memory 1 address register
edma_s7fctrl	EDMA stream 7 FIFO control register
edma_s8ctrl	EDMA stream 8 control register
edma_s8dtcnt	EDMA stream 8 number of data register
edma_s8paddr	EDMA stream 8 peripheral address register
edma_s8m0addr	EDMA stream 8 memory 0 address register
edma_s8m1addr	EDMA stream 8 memory 1 address register
edma_s8fctrl	EDMA stream 8 FIFO control register
edma_llctrl	EDMA linked table control register
edma_s1llp	EDMA stream 1 linked table pointer register
edma_s2llp	EDMA stream 2 linked table pointer register
edma_s3llp	EDMA stream 3 linked table pointer register
edma_s4llp	EDMA stream 4 linked table pointer register
edma_s5llp	EDMA stream 5 linked table pointer register
edma_s6llp	EDMA stream 6 linked table pointer register
edma_s7llp	EDMA stream 7 linked table pointer register
edma_s8llp	EDMA stream 8 linked table pointer register
edma_s2dctrl	EDMA 2D transfer control register
edma_s12dcnt	EDMA stream 1 2D transfer count register
edma_s1stride	EDMA stream 1 2D transfer stride register
edma_s22dcnt	EDMA stream 2 2D transfer count register
edma_s2stride	EDMA stream 2 2D transfer stride register
edma_s32dcnt	EDMA stream 3 2D transfer count register
edma_s3stride	EDMA stream 3 2D transfer stride register
edma_s42dcnt	EDMA stream 4 2D transfer count register
edma_s4stride	EDMA stream 4 2D transfer stride register
edma_s52dcnt	EDMA stream 5 2D transfer count register
edma_s5stride	EDMA stream 5 2D transfer stride register
edma_s62dcnt	EDMA stream 6 2D transfer count register
edma_s6stride	EDMA stream 6 2D transfer stride register

Register	Description
edma_s72dcnt	EDMA stream 7 2D transfer count register
edma_s7stride	EDMA stream 7 2D transfer stride register
edma_s82dcnt	EDMA stream 8 2D transfer count register
edma_s8stride	EDMA stream 8 2D transfer stride register
edma_muxsel	EDMAMUX enable register
edma_muxc1ctrl	EDMAMUX channel 1 control register
edma_muxc2ctrl	EDMAMUX channel 2 control register
edma_muxc3ctrl	EDMAMUX channel 3 control register
edma_muxc4ctrl	EDMAMUX channel 4 control register
edma_muxc5ctrl	EDMAMUX channel 5 control register
edma_muxc6ctrl	EDMAMUX channel 6 control register
edma_muxc7ctrl	EDMAMUX channel 7 control register
edma_muxg1ctrl	EDMAMUX request generator 1 control register
edma_muxg2ctrl	EDMAMUX request generator 2 control register
edma_muxg3ctrl	EDMAMUX request generator 3 control register
edma_muxg4ctrl	EDMAMUX request generator 4 control register
edma_muxsyncsts	EDMAMUX synchronization status register
edma_muxsyncclr	EDMAMUX synchronization status clear register
edma_muxgsts	EDMAMUX request generator status register
edma_muxgclr	EDMAMUX request generator status clear register

5.10.1 edma_reset function

The table below describes the function edma_reset.

Table 234. edma_reset function

Name	Description
Function name	edma_reset
Function prototype	void edma_reset(edma_stream_type *edma_streamx);
Function description	Reset the selected EDMA stream
Input parameter 1	edma_streamx: EDMA stream, x=1...8
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* reset edma stream1 */
edma_reset (EDMA_STREAM1);
```

5.10.2 edma_init function

The table below describes the function edma_init.

Table 235. edma_init function function

Name	Description
Function name	edma_init
Function prototype	void edma_init(edma_stream_type *edma_streamx, edma_init_type *edma_init_struct);
Function description	Initialize the selected EDMA stream
Input parameter 1	edma_streamx: EDMA stream, x=1...8
Input parameter 2	edma_init_struct: edma_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

edma_init_type structure

The edma_init_type is defined in the at32f435_437_edma.h.

```
typedef struct
{
    uint32_t                    peripheral_base_addr;
    uint32_t                    memory0_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;
    confirm_state               fifo_mode_enable;
    edma_fifo_threshold_type    fifo_threshold;
    edma_memory_burst_type      memory_burst_mode;
    edma_peripheral_burst_type  peripheral_burst_mode;
} edma_init_type;
```

peripheral_base_addr

Set the peripheral address of EDMA data stream.

memory_base_addr

Set the memory address of EDMA data stream.

direction

Set the transfer direction of EDMA data stream.

EDMA_DIR_PERIPHERAL_TO_MEMORY: Peripheral to memory
EDMA_DIR_MEMORY_TO_PERIPHERAL: Memory to peripheral
EDMA_DIR_MEMORY_TO_MEMORY: Memory to memory

buffer_size

Set the number of data transfer of a EDMA channel.

peripheral_inc_enable

Enable/disable EDMA data stream peripheral address auto increment
FALSE: Peripheral address is not incremented
TRUE: Peripheral address is incremented

memory_inc_enable

Enable/disable EDMA data stream memory address auto increment
FALSE: Memory address is not incremented
TRUE: Memory address is incremented

peripheral_data_width

Set EDMA peripheral data width

EDMA_PERIPHERAL_DATA_WIDTH_BYTE:	Byte
EDMA_PERIPHERAL_DATA_WIDTH_HALFWORD:	Half-word
EDMA_PERIPHERAL_DATA_WIDTH_WORD:	Word

memory_data_width

Set EDMA memory data width

EDMA_MEMORY_DATA_WIDTH_BYTE:	Byte
EDMA_MEMORY_DATA_WIDTH_HALFWORD:	Half-word
EDMA_MEMORY_DATA_WIDTH_WORD:	Word

loop_mode_enable

Set EDMA loop mode.
FALSE: EDMA single mode
TRUE: EDMA loop mode

priority

Set EDMA data stream priority.

EDMA_PRIORITY_LOW:	Low
EDMA_PRIORITY_MEDIUM:	Medium
EDMA_PRIORITY_HIGH:	High
EDMA_PRIORITY_VERY_HIGH:	Very high

fifo_mode_enable

Enable/disable EDMA data stream FIFO mode.
FALSE: Direct mode
TRUE: FIFO mode

fifo_threshold

Set EDMA data stream FIFO threshold.

EDMA_FIFO_THRESHOLD_1QUARTER:	1/4 FIFO
EDMA_FIFO_THRESHOLD_HALF:	1/2 FIFO
EDMA_FIFO_THRESHOLD_3QUARTER:	3/4 FIFO
EDMA_FIFO_THRESHOLD_FULL:	Full FIFO

memory_burst_mode

Enable/disable EDMA memory burst mode.

EDMA_MEMORY_SINGLE:	Memory burst disabled
EDMA_MEMORY_BURST_4:	Memory burst enabled, transfer 4 data each time
EDMA_MEMORY_BURST_8:	Memory burst enabled, transfer 8 data each time
EDMA_MEMORY_BURST_16:	Memory burst enabled, transfer 16 data each time

peripheral_burst_mode

Enable/disable EDMA peripheral burst mode.

EDMA_PERIPHERAL_SINGLE:	Peripheral burst disabled

EDMA_PERIPHERAL_BURST_4: Peripheral burst enabled, transfer 4 data each time
EDMA_PERIPHERAL_BURST_8: Peripheral burst enabled, transfer 8 data each time
EDMA_PERIPHERAL_BURST_16: Peripheral burst enabled, transfer 16 data each time

Example:

```
edmamux_sync_init_type    edmamux_sync_init_struct;
/* edma stream4 configuration */
edma_init_struct.direction = EDMA_DIR_PERIPHERAL_TO_MEMORY;
edma_init_struct.buffer_size = (uint32_t)16;
edma_init_struct.peripheral_data_width = EDMA_PERIPHERAL_DATA_WIDTH_HALFWORD;
edma_init_struct.peripheral_base_addr = (uint32_t)src_buffer;
edma_init_struct.peripheral_inc_enable = TRUE;
edma_init_struct.memory_data_width = EDMA_MEMORY_DATA_WIDTH_HALFWORD;
edma_init_struct.memory0_base_addr = (uint32_t)dst_buffer;
edma_init_struct.memory_inc_enable = TRUE;
edma_init_struct.peripheral_burst_mode = EDMA_PERIPHERAL_SINGLE;
edma_init_struct.memory_burst_mode = EDMA_MEMORY_SINGLE;
edma_init_struct.fifo_mode_enable = FALSE;
edma_init_struct.fifo_threshold = EDMA_FIFO_THRESHOLD_1QUARTER;
edma_init_struct.priority = EDMA_PRIORITY_HIGH;
edma_init_struct.loop_mode_enable = FALSE;
edma_init(EDMA_STREAM4, &edma_init_struct);
```

5.10.3 edma_default_para_init function

The table below describes the function edma_default_para_init.

Table 236. edma_default_para_init function

Name	Description
Function name	edma_default_para_init
Function prototype	void edma_default_para_init(edma_init_type *edma_init_struct);
Function description	Initialize the parameters of the edma_init_struct
Input parameter 1	edma_init_struct: edma_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of the edma_init_struct members.

Table 237. edma_init_struct default values

Member	Default value
peripheral_base_addr	0x0
memory_base_addr	0x0
direction	EDMA_DIR_PERIPHERAL_TO_MEMORY
buffer_size	0x0
peripheral_inc_enable	FALSE
memory_inc_enable	FALSE
peripheral_data_width	EDMA_PERIPHERAL_DATA_WIDTH_BYTE

Member	Default value
memory0_data_width	EDMA_MEMORY_DATA_WIDTH_BYTE
loop_mode_enable	FALSE
priority	EDMA_PRIORITY_LOW
fifo_mode_enable	FALSE
fifo_threshold	EDMA_FIFO_THRESHOLD_1QUARTER
memory_burst_mode	EDMA_MEMORY_SINGLE
peripheral_burst_mode	EDMA_PERIPHERAL_SINGLE

Example:

```
/* edma init config with its default value */
edmamux_sync_init_type    edmamux_sync_init_struct;
edma_default_para_init(&edma_init_struct);
```

5.10.4 edma_stream_enable function

The table below describes the function edma_stream_enable.

Table 238. edma_stream_enable function

Name	Description
Function name	edma_stream_enable
Function prototype	void edma_stream_enable(edma_stream_type *edma_streamx, confirm_state new_state);
Function description	Enable the selected EDMA stream
Input parameter 1	edma_streamx: EDMA stream, x=1...8
Input parameter 2	new_state: Enable/disable stream
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

Enable/disable EDMA data stream.

FALSE: Disabled

TRUE: Enabled

Example:

```
/* enable edma stream4 */
edma_stream_enable(EDMA_STREAM4, TRUE);
```

5.10.5 edma_interrupt_enable function

The table below describes the function edma_interrupt_enable.

Table 239. edma_interrupt_enable function

Name	Description
Function name	edma_interrupt_enable
Function prototype	void edma_interrupt_enable(edma_stream_type *edma_streamx, uint32_t edma_int, confirm_state new_state);
Function description	Enable the selected EDMA stream interrupt
Input parameter 1	edma_streamx: EDMA stream, x=1...8
Input parameter 2	edma_int: selected interrupt source
Input parameter 3	new_state: enable or disable interrupt
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

edma_int

Select EDMA stream interrupt source.

EDMA_FDT_INT: Transfer complete interrupt
EDMA_HDT_INT: Half transfer complete interrupt
EDMA_DTERR_INT: Transfer error interrupt
EDMA_DMERR_INT: Direct mode error interrupt
EDMA_FERR_INT: FIFO error interrupt

new_state

Enable/disable DMA channel interrupt.

FALSE: Interrupt disabled
TRUE: Interrupt enabled

Example:

```
/* enable transfer full data intterrupt */
edma_interrupt_enable(EDMA_STREAM4, EDMA_FDT_INT, TRUE);
```

5.10.6 edma_peripheral_inc_offset_set function

The table below describes the function edma_peripheral_inc_offset_set.

Table 240. edma_peripheral_inc_offset_set function

Name	Description
Function name	edma_peripheral_inc_offset_set
Function prototype	void edma_peripheral_inc_offset_set(edma_stream_type *edma_streamx, edma_peripheral_inc_offset_type offset);
Function description	Configure peripheral address auto offset mode
Input parameter 1	edma_streamx: EDMA stream, x=1...8
Input parameter 2	offset: auto offset mode,
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

offset

Select auto offset mode.

EDMA_PERIPHERAL_INC_PSIZE: Offset is determined by psice

EDMA_PERIPHERAL_INC_4_BYTE: Auto offset of 4 bytes

Example:

```
/* cofig peripheral offset */
edma_peripheral_inc_offset_set (EDMA_STREAM4, EDMA_PERIPHERAL_INC_4_BYTE);
```

5.10.7 edma_flow_controller_enable

The table below describes the function edma_flow_controller_enable

Table 241. edma_flow_controller_enable function

Name	Description
Function name	edma_flow_controller_enable
Function prototype	void edma_flow_controller_enable(edma_stream_type *edma_streamx, confirm_state new_state);
Function description	Enable EDMA peripheral flow control
Input parameter 1	edma_streamx: EDMA stream, x=1...8
Input parameter 2	new_state: enable/disable peripheral flow control
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

Enable or disable EDMA channel interrupt.

FALSE: Interrupt disabled

TRUE: Interrupt enabled

Example:

```
/* enable stream4 peripheral stream control */
edma_flow_controller_enable (EDMA_STREAM4, TRUE);
```

5.10.8 edma_data_number_set function

The table below describes the function edma_data_number_set

Table 242. edma_data_number_set function

Name	Description
Function name	edma_data_number_set
Function prototype	void edma_data_number_set(edma_stream_type *edma_streamx, uint16_t data_number);
Function description	Configure the EDMA stream-x number of data register
Input parameter 1	edma_streamx: EDMA stream, x=1...8
Input parameter 2	data_number: data transfer size, maximum: 65535
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set edma stream1 data count is 0x100*/
edma_data_number_set(EDMA_STREAM1, 0x100);
```

5.10.9 edma_data_number_get function

The table below describes the function edma_data_number_get.

Table 243. edma_data_number_get function

Name	Description
Function name	edma_data_number_get
Function prototype	uint16_t edma_data_number_get(edma_stream_type *edma_streamx);
Function description	Get the value of EDMA stream-x number of data register
Input parameter 1	edma_streamx: EDMA stream, x=1...8
Output parameter	NA
Return value	Return the number of data transfer
Required preconditions	NA
Called functions	NA

Example:

```
/* get edma stream1 data count*/
uint16_t data_counter;
data_counter = edma_data_number_get(EDMA_STREAM1);
```

5.10.10 edma_double_buffer_mode_init function

The table below describes the function edma_double_buffer_mode_init.

Table 244. edma_double_buffer_mode_init function

Name	Description
Function name	edma_double_buffer_mode_init
Function prototype	void edma_double_buffer_mode_init(edma_stream_type *edma_streamx, uint32_t memory1_addr, edma_memory_type current_memory);
Function description	Configure the EDMA stream dual buffer mode
Input parameter 1	edma_streamx: EDMA stream, x=1...8
Input parameter 2	memory1_addr: memory0 address
Input parameter 3	current_memory: memory0 or memory1
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* stream4 double buffer mode config */
edma_double_buffer_mode_init (EDMA_STREAM4, 0x20000400; EDMA_MEMORY_0);
```

5.10.11 edma_double_buffer_mode_enable function

The table below describes the function edma_double_buffer_mode_enable.

Table 245. edma_double_buffer_mode_enable function

Name	Description
Function name	edma_double_buffer_mode_enable
Function prototype	void edma_double_buffer_mode_enable(edma_stream_type *edma_streamx, confirm_state new_state);
Function description	Enable EDMA stream dual buffer mode
Input parameter 1	edma_streamx: EDMA stream, x=1...8
Input parameter 2	new_state: Enable/disable dual buffer mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* stream4 double buffer mode enable */
edma_double_buffer_mode_enable (EDMA_STREAM4, TRUE);
```

5.10.12 edma_memory_addr_set function

The table below describes the function edma_memory_addr_set.

Table 246. edma_memory_addr_set function

Name	Description
Function name	edma_memory_addr_set
Function prototype	void edma_memory_addr_set(edma_stream_type *edma_streamx, uint32_t memory_addr, uint32_t memory_target);
Function description	Configure the EDMA stream memory address in dual buffer mode
Input parameter 1	edma_streamx: EDMA stream, x=1...8
Input parameter 2	memory_addr: memory address
Input parameter 2	memory_target: the target memory
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set edma stream1 memory1 adddress */
edma_memory_addr_set (EDMA_STREAM1, 0x20000400, EDMA_MEMORY_1);
```

5.10.13 edma_stream_status_get function

The table below describes the function edma_stream_status_get.

Table 247. edma_stream_status_get function

Name	Description
Function name	edma_stream_status_get
Function prototype	flag_status edma_stream_status_get(edma_stream_type *edma_streamx);
Function description	Get the EDMA stream status
Input parameter 1	edma_streamx: EDMA stream, x=1...8
Output parameter	NA
Return value	Return the EDMA stream status
Required preconditions	NA
Called functions	NA

Example:

```
/* get edma stream1 status*/
flag_status sts;
sts = edma_stream_status_get(EDMA_STREAM1);
```

5.10.14 edma_fifo_status_get function

The table below describes the function edma_fifo_status_get.

Table 248. edma_fifo_status_get function

Name	Description
Function name	edma_fifo_status_get
Function prototype	uint8_t edma_fifo_status_get(edma_stream_type *edma_streamx);
Function description	Get the EDMA stream FIFO status
Input parameter 1	edma_streamx: EDMA stream, x=1...8
Output parameter	NA
Return value	Return the EDMA stream FIFO status
Required preconditions	NA
Called functions	NA

Example:

```
/* get edma stream1 fifo status*/
uint8_t fifosts;
fifosts = edma_fifo_status_get(EDMA_STREAM1);
```

5.10.15 edma_flag_get function

The table below describes the function edma_flag_get.

Table 249. edma_flag_get function

Name	Description
Function name	edma_flag_get
Function prototype	flag_status edma_flag_get(uint32_t edma_flag);
Function description	Get the EDMA stream status flag
Input parameter 1	*edma_flag* : the desired status flag
Output parameter	NA
Return value	Return the EDMA stream status flag
Required preconditions	NA
Called functions	NA

edma_flag

The edma_flag is used to get a status flag, including:

EDMA_FERR1_FLAG: Data stream 1 FIFO error flag
EDMA_DMERR1_FLAG: Data stream 1 dual buffer error flag
EDMA_DTERR1_FLAG: Data stream 1 direct mode error flag
EDMA_HDT1_FLAG: Data stream 1 half transfer complete flag
EDMA_FDT1_FLAG: Data stream 1 transfer complete flag
EDMA_FERR2_FLAG: Data stream 2 FIFO error flag
EDMA_DMERR2_FLAG: Data stream 2 dual buffer error flag
EDMA_DTERR2_FLAG: Data stream 2 direct mode error flag
EDMA_HDT2_FLAG: Data stream 2 half transfer complete flag
EDMA_FDT2_FLAG: Data stream 2 transfer complete flag
EDMA_FERR3_FLAG: Data stream 3 FIFO error flag

EDMA_DMERR3_FLAG:	Data stream 3 dual buffer error flag
EDMA_DTERR3_FLAG:	Data stream 3 direct mode error flag
EDMA_HDT3_FLAG:	Data stream 3 half transfer complete flag
EDMA_FDT3_FLAG:	Data stream 3 transfer complete flag
EDMA_FERR4_FLAG:	Data stream 4 FIFO error flag
EDMA_DMERR4_FLAG:	Data stream 4 dual buffer error flag
EDMA_DTERR4_FLAG:	Data stream 4 direct mode error flag
EDMA_HDT4_FLAG:	Data stream 4 half transfer complete flag
EDMA_FDT4_FLAG:	Data stream 4 transfer complete flag
EDMA_FERR5_FLAG:	Data stream 5 FIFO error flag
EDMA_DMERR5_FLAG:	Data stream 5 dual buffer error flag
EDMA_DTERR5_FLAG:	Data stream 5 direct mode error flag
EDMA_HDT5_FLAG:	Data stream 5 half transfer complete flag
EDMA_FDT5_FLAG:	Data stream 5 transfer complete flag
EDMA_FERR6_FLAG:	Data stream 6 FIFO error flag
EDMA_DMERR6_FLAG:	Data stream 6 dual buffer error flag
EDMA_DTERR6_FLAG:	Data stream 6 direct mode error flag
EDMA_HDT6_FLAG:	Data stream 6 half transfer complete flag
EDMA_FDT6_FLAG:	Data stream 6 transfer complete flag
EDMA_FERR7_FLAG:	Data stream 7 FIFO error flag
EDMA_DMERR7_FLAG:	Data stream 7 dual buffer error flag
EDMA_DTERR7_FLAG:	Data stream 7 direct mode error flag
EDMA_HDT7_FLAG:	Data stream 7 half transfer complete flag
EDMA_FDT7_FLAG:	Data stream 7 transfer complete flag
EDMA_FERR8_FLAG:	Data stream 8 FIFO error flag
EDMA_DMERR8_FLAG:	Data stream 8 dual buffer error flag
EDMA_DTERR8_FLAG:	Data stream 8 direct mode error flag
EDMA_HDT8_FLAG:	Data stream 8 half transfer complete flag
EDMA_FDT8_FLAG:	Data stream 8 transfer complete flag

Example:

```
/* get edma stream1 full data transfer flag dstatus*/
uint8_t sts;
sts = edma_flag_get(EDMA_FDT1_FLAG);
```

5.10.16 edma_2d_init function

The table below describes the function edma_2d_init.

Table 250. edma_2d_init function

Name	Description
Function name	edma_2d_init
Function prototype	void edma_2d_init(edma_stream_2d_type *edma_streamx_2d, int16_t src_stride, int16_t dst_stride, uint16_t xcnt, uint16_t ycnt);
Function description	Configure EDMA stream 2D transfer mode
Input parameter 1	*edma_streamx_2d* : 2D transfer EDMA stream, x=1...8
Input parameter 2	src_stride: source stride
Input parameter 3	dst_stride: destination stride
Input parameter 4	xcnt: x-axis count
Input parameter 5	ycnt: y-axis count
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

edma_streamx_2d

The edma_streamx_2d is used to select a status flag, including:

EDMA_STREAM1_2D

EDMA_STREAM2_2D

EDMA_STREAM3_2D

EDMA_STREAM4_2D

EDMA_STREAM5_2D

EDMA_STREAM6_2D

EDMA_STREAM7_2D

EDMA_STREAM8_2D

Example:

```
/* stream4 2d transfer mode config */
edma_2d_init (EDMA_STREAM4_2D, 0x00; 0x00; 0x04; 0x10);
```

5.10.17 edma_2d_enable function

The table below describes the function edma_2d_enable.

Table 251. edma_2d_enable function

Name	Description
Function name	edma_2d_enable
Function prototype	void edma_2d_enable(edma_stream_2d_type *edma_streamx_2d, confirm_state new_state);
Function description	Enable EDMA stream 2D transfer mode
Input parameter 1	*edma_streamx_2d* : 2D transfer EDMA stream, x=1...8
Input parameter 2	new_state: enable/disable 2D transfer mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* stream4 2d transfer mode enable */
edma_2d_enable (EDMA_STREAM4_2D, TRUE);
```

5.10.18 edma_link_list_init function

The table below describes the function edma_link_list_init.

Table 252. edma_link_list_init function

Name	Description
Function name	edma_link_list_init
Function prototype	void edma_link_list_init(edma_stream_link_list_type *edma_streamx_ll, uint32_t pointer);
Function description	Initialize EDMA linked table transfer
Input parameter 1	*edma_streamx_ll* : EDMA stream, x=1...8
Input parameter 2	pointer: linked table transfer descriptor storage pointer address
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

edma_streamx_ll

The edma_streamx_ll is used to select a status flag, including

EDMA_STREAM1_ll

EDMA_STREAM2_ll

EDMA_STREAM3_ll

EDMA_STREAM4_ll

EDMA_STREAM5_ll

EDMA_STREAM6_ll

EDMA_STREAM7_ll

EDMA_STREAM8_ll

Example:

```
/* link list descriptor array, start address must be aligned by 16 bytes */
__ALIGNED(16) ll_descriptors_type edma_ll_descriptors_tx[LIST_COUNT];
/* edma stream1 link list mode configuration */
edma_link_list_init(EDMA_STREAM1_LL, (uint32_t)edma_ll_descriptors_tx);
```

5.10.19 edma_link_list_enable function

The table below describes the function edma_link_list_enable.

Table 253. edma_link_list_enable function

Name	Description
Function name	edma_link_list_enable
Function prototype	void edma_link_list_enable(edma_stream_link_list_type *edma_streamx_ll, confirm_state new_state);
Function description	Enable EDMA stream linked table transfer
Input parameter 1	*edma_streamx_ll* : EDMA stream, x=1...8
Input parameter 2	new_state: enable/disable 2D transfer mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* edma stream1 link list mode enable */
void edma_link_list_enable(edma_stream_link_list_type *edma_streamx_ll, confirm_state new_state);
```

5.10.20 edma_flag_clear function

The table below describes the function edma_flag_clear.

Table 254. edma_flag_clear function

Name	Description
Function name	edma_flag_clear
Function prototype	void edma_flag_clear(uint32_t edma_flag);
Function description	Get the EDMA stream status flag
Input parameter 1	*edma_flag* : desired status flag
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* clear edma stream1 full data transfer flag dstatus*/
edma_flag_get(EDMA_FDT1_FLAG);
```

The The table below describes the default values of the dmamux_sync_init_struct members.

Table 255. dmamux_sync_init_struct values

Member	Default value
sync_enable	FALSE
sync_event_enable	FALSE
sync_polarity	DMAMUX_SYNC_POLARITY_DISABLE
sync_request_number	0x0
sync_signal_sel	(dmamux_sync_id_sel_type)0

Example:

```
/* dmamux sync init config with its default value */
dmamux_sync_init_type dmamux_sync_init_struct = {0};
dmamux_sync_default_para_init (&dmamux_sync_init_struct);
```

5.10.21 edmamux_enable function

The table below describes the function edmamux_enable.

Table 256. edmamux_enable function

Name	Description
Function name	edmamux_enable
Function prototype	void edmamux_enable(confirm_state new_state);
Function description	Enable EDMAMUX feature
Input parameter 1	new_state: enable or disable channel
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

Enable or disable DMA channel.

FALSE: Channel disabled

TRUE: Channel enabled

Example:

```
/* edmamux function enable */
edmamux_enable(TRUE);
```

5.10.22 edmamux_init function

The table below describes the function edmamux_init.

Table 257. edmamux_init function

Name	Description
Function name	edmamux_init
Function prototype	void edmamux_init(edmamux_channel_type *edmamux_channelx, edmamux_requst_id_sel_type edmamux_req_id);
Function description	Configure EDMAMUX
Input parameter 1	*edmamux_channelx* : EDMAMUX channel, x=1…8
Input parameter 2	*edmamux_req_id* : DMAMUX channel request ID
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

edmamux_channelx

This is used to select a DMAMUX channel, including:

EDMAMUX_CHANNEL1

EDMAMUX_CHANNEL2

EDMAMUX_CHANNEL3

EDMAMUX_CHANNEL4

EDMAMUX_CHANNEL5

EDMAMUX_CHANNEL6

EDMAMUX_CHANNEL7

EDMAMUX_CHANNEL8

edmamux_req_id

The table below describes the EDMAMUX channel request ID.

Table 258. EDMAMUX channel request source ID

Request source ID	Description	Request source ID	Description
0x01	EDMAMUX_DMAREQ_ID_REQ_G1	0x02	EDMAMUX_DMAREQ_ID_REQ_G2
0x03	EDMAMUX_DMAREQ_ID_REQ_G3	0x04	EDMAMUX_DMAREQ_ID_REQ_G4
0x05	EDMAMUX_DMAREQ_ID_ADC1	0x24	EDMAMUX_DMAREQ_ID_ADC2
0x25	EDMAMUX_DMAREQ_ID_ADC3	0x06	EDMAMUX_DMAREQ_ID_DAC1
0x29	EDMAMUX_DMAREQ_ID_DAC2	0x08	EDMAMUX_DMAREQ_ID_TMR6_OVERFLOW
0x09	EDMAMUX_DMAREQ_ID_TMR7_OVERFLOW	0x0A	EDMAMUX_DMAREQ_ID_SPI1_RX
0x0B	EDMAMUX_DMAREQ_ID_SPI1_TX	0x0C	EDMAMUX_DMAREQ_ID_SPI2_RX
0x0D	EDMAMUX_DMAREQ_ID_SPI2_TX	0x0E	EDMAMUX_DMAREQ_ID_SPI3_RX
0x0F	EDMAMUX_DMAREQ_ID_SPI3_TX	0x6A	EDMAMUX_DMAREQ_ID_SPI4_RX
0x6B	EDMAMUX_DMAREQ_ID_SPI4_TX	0x6E	EDMAMUX_DMAREQ_ID_I2S2_EXT_RX
0x6F	EDMAMUX_DMAREQ_ID_I2S2_EXT_TX	0x70	EDMAMUX_DMAREQ_ID_I2S3_EXT_RX
0x71	EDMAMUX_DMAREQ_ID_I2S3_EXT_TX	0x10	EDMAMUX_DMAREQ_ID_I2C1_RX

Request source ID	Description	Request source ID	Description
0x11	EDMAMUX_DMAREQ_ID_I2C1_TX	0x12	EDMAMUX_DMAREQ_ID_I2C2_RX
0x13	EDMAMUX_DMAREQ_ID_I2C2_TX	0x14	EDMAMUX_DMAREQ_ID_I2C3_RX
0x15	EDMAMUX_DMAREQ_ID_I2C3_TX	0x18	EDMAMUX_DMAREQ_ID_USART1_RX
0x19	EDMAMUX_DMAREQ_ID_USART1_TX	0x1A	EDMAMUX_DMAREQ_ID_USART2_RX
0x1B	EDMAMUX_DMAREQ_ID_USART2_TX	0x1C	EDMAMUX_DMAREQ_ID_USART3_RX
0x1D	EDMAMUX_DMAREQ_ID_USART3_TX	0x1E	EDMAMUX_DMAREQ_ID_UART4_RX
0x1F	EDMAMUX_DMAREQ_ID_UART4_TX	0x20	EDMAMUX_DMAREQ_ID_UART5_RX
0x21	EDMAMUX_DMAREQ_ID_UART5_TX	0x72	EDMAMUX_DMAREQ_ID_USART6_RX
0x73	EDMAMUX_DMAREQ_ID_USART6_TX	0x74	EDMAMUX_DMAREQ_ID_UART7_RX
0x75	EDMAMUX_DMAREQ_ID_UART7_TX	0x76	EDMAMUX_DMAREQ_ID_UART8_RX
0x77	EDMAMUX_DMAREQ_ID_UART8_TX	0x27	EDMAMUX_DMAREQ_ID_SDIO1
0x67	EDMAMUX_DMAREQ_ID_SDIO2	0x28	EDMAMUX_DMAREQ_ID_QSPI1
0x68	EDMAMUX_DMAREQ_ID_QSPI2	0x2A	EDMAMUX_DMAREQ_ID_TMR1_CH1
0x2B	EDMAMUX_DMAREQ_ID_TMR1_CH2	0x2C	EDMAMUX_DMAREQ_ID_TMR1_CH3
0x2D	EDMAMUX_DMAREQ_ID_TMR1_CH4	0x2E	EDMAMUX_DMAREQ_ID_TMR1_OVERFLOW
0x2F	EDMAMUX_DMAREQ_ID_TMR1_TRIG	0x30	EDMAMUX_DMAREQ_ID_TMR1_HALL
0x31	EDMAMUX_DMAREQ_ID_TMR8_CH1	0x32	EDMAMUX_DMAREQ_ID_TMR8_CH2
0x33	EDMAMUX_DMAREQ_ID_TMR8_CH3	0x34	EDMAMUX_DMAREQ_ID_TMR8_CH4
0x35	EDMAMUX_DMAREQ_ID_TMR8_OVERFLOW	0x36	EDMAMUX_DMAREQ_ID_TMR8_TRIG
0x37	EDMAMUX_DMAREQ_ID_TMR8_HALL	0x38	EDMAMUX_DMAREQ_ID_TMR2_CH1
0x39	EDMAMUX_DMAREQ_ID_TMR2_CH2	0x3A	EDMAMUX_DMAREQ_ID_TMR2_CH3
0x3B	EDMAMUX_DMAREQ_ID_TMR2_CH4	0x3C	EDMAMUX_DMAREQ_ID_TMR2_OVERFLOW
0x7E	EDMAMUX_DMAREQ_ID_TMR2_TRIG	0x3D	EDMAMUX_DMAREQ_ID_TMR3_CH1
0x3E	EDMAMUX_DMAREQ_ID_TMR3_CH2	0x3F	EDMAMUX_DMAREQ_ID_TMR3_CH3
0x40	EDMAMUX_DMAREQ_ID_TMR3_CH4	0x41	EDMAMUX_DMAREQ_ID_TMR3_OVERFLOW
0x42	EDMAMUX_DMAREQ_ID_TMR3_TRIG	0x43	EDMAMUX_DMAREQ_ID_TMR4_CH1
0x44	EDMAMUX_DMAREQ_ID_TMR4_CH2	0x45	EDMAMUX_DMAREQ_ID_TMR4_CH3
0x46	EDMAMUX_DMAREQ_ID_TMR4_CH4	0x47	EDMAMUX_DMAREQ_ID_TMR4_OVERFLOW
0x7F	EDMAMUX_DMAREQ_ID_TMR4_TRIG	0x48	EDMAMUX_DMAREQ_ID_TMR5_CH1
0x49	EDMAMUX_DMAREQ_ID_TMR5_CH2	0x4A	EDMAMUX_DMAREQ_ID_TMR5_CH3
0x4B	EDMAMUX_DMAREQ_ID_TMR5_CH4	0x4C	EDMAMUX_DMAREQ_ID_TMR5_OVERFLOW
0x4D	EDMAMUX_DMAREQ_ID_TMR5_TRIG	0x56	EDMAMUX_DMAREQ_ID_TMR20_CH1
0x57	EDMAMUX_DMAREQ_ID_TMR20_CH2	0x58	EDMAMUX_DMAREQ_ID_TMR20_CH3
0x59	EDMAMUX_DMAREQ_ID_TMR20_CH4	0x5A	EDMAMUX_DMAREQ_ID_TMR20_OVERFLOW
0x5D	EDMAMUX_DMAREQ_ID_TMR20_TRIG	0x5E	EDMAMUX_DMAREQ_ID_TMR20_HALL
0x69	EDMAMUX_DMAREQ_ID_DVP		

Example:

```
/* generator1 for edmamux channel4 as dma request */
edmamux_init(EDMAMUX_CHANNEL4, EDMAMUX_DMAREQ_ID_REQ_G1);
```

5.10.23 edmamux_sync_default_para_init function

The table below describes the function edmamux_sync_default_para_init.

Table 259. edmamux_sync_default_para_init function

Name	Description
Function name	edmamux_sync_default_para_init
Function prototype	void edmamux_sync_default_para_init(edmamux_sync_init_type *edmamux_sync_init_struct);
Function description	Initialize parameters in the edmamux_sync_init_struct
Input parameter 1	edmamux_sync_init_struct: edmamux_sync_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of the edmamux_sync_init_struct members.

Table 260. edmamux_sync_init_struct default values

Member	Default values
sync_enable	FALSE
sync_event_enable	FALSE
sync_polarity	EDMAMUX_SYNC_POLARITY_DISABLE
sync_request_number	0x0
sync_signal_sel	(edmamux_sync_id_sel_type)0

Example:

```
/* edmamux sync init config with its default value */
edmamux_sync_init_type edmamux_sync_init_struct = {0};
edmamux_sync_default_para_init (&edmamux_sync_init_struct);
```

5.10.24 edmamux_sync_config function

The table below describes the function edmamux_sync_config.

Table 261. edmamux_sync_config function

Name	Description
Function name	edmamux_sync_config
Function prototype	void edmamux_sync_config(edmamux_channel_type * edmamux_channelx, edmamux_sync_init_type *edmamux_sync_init_struct);
Function description	Configure the EDMAMUX synchronous module
Input parameter 1	*edmamux_channelx* DMAMUX channel, x=1…7
Input parameter 2	edmamux _sync_init_struct: edmamux _sync_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

edmamux_sync_init_struct

The edmamux _sync_init_type is defined in the at32f435_437_dma.h.

```
typedef struct
{
    dmamux_sync_id_sel_type     sync_signal_sel;
    uint32_t                    sync_polarity;
    uint32_t                    sync_request_number;
    confirm_state               sync_event_enable;
    confirm_state               sync_enable;
} edmamux_sync_init_type;
```

sync_signal_sel

Select signal source for synchronous module.

EDMAMUX_SYNC_ID_EXINT0: External exint0 signal
EDMAMUX_SYNC_ID_EXINT1: External exint1 signal
EDMAMUX_SYNC_ID_EXINT2: External exint2 signal
EDMAMUX_SYNC_ID_EXINT3: External exint3 signal
EDMAMUX_SYNC_ID_EXINT4: External exint4 signal
EDMAMUX_SYNC_ID_EXINT5: External exint5 signal
EDMAMUX_SYNC_ID_EXINT6: External exint6 signal
EDMAMUX_SYNC_ID_EXINT7: External exint7 signal
EDMAMUX_SYNC_ID_EXINT8: External exint8 signal
EDMAMUX_SYNC_ID_EXINT9: External exint9 signal
EDMAMUX_SYNC_ID_EXINT10: External exint10 signal
EDMAMUX_SYNC_ID_EXINT11: External exint11 signal
EDMAMUX_SYNC_ID_EXINT12: External exint12 signal
EDMAMUX_SYNC_ID_EXINT13: External exint13 signal
EDMAMUX_SYNC_ID_EXINT14: External exint14 signal
EDMAMUX_SYNC_ID_EXINT15: External exint15 signal
EDMAMUX_SYNC_ID_DMAMUX_CH1_EVT: dmamux channel 1 event
EDMAMUX_SYNC_ID_DMAMUX_CH2_EVT: dmamux channel 2 event
EDMAMUX_SYNC_ID_DMAMUX_CH3_EVT: dmamux channel 3 event
EDMAMUX_SYNC_ID_DMAMUX_CH4_EVT: dmamux channel 4 event
EDMAMUX_SYNC_ID_DMAMUX_CH5_EVT: dmamux channel 5 event
EDMAMUX_SYNC_ID_DMAMUX_CH6_EVT: dmamux channel 6 event
EDMAMUX_SYNC_ID_DMAMUX_CH7_EVT: dmamux channel 7 event

sync_polarity

Select the signal polarity for synchronous module.

EDMAMUX_SYNC_POLARITY_RISING: Rising edge
EDMAMUX_SYNC_POLARITY_FALLING: Falling edge
EDMAMUX_SYNC_POLARITY_RISING_FALLING: Rising edge and falling edge

sync_request_number

The number of DMA requests that can be synchronized

Range: 1~32

sync_event_enable

Generation of synchronous event

TURE: Synchronous event is generated
FALSE: Synchronous event is not generated

sync_enable

Enable or disable synchronous module

FALSE: Synchronous module disabled

TRUE: Synchronous module enabled

Example:

```
edmamux_sync_default_para_init(&edmamux_sync_init_struct);
edmamux_sync_init_struct.sync_request_number = 4;
edmamux_sync_init_struct.sync_signal_sel = DMAMUX_SYNC_ID_EXINT1;
edmamux_sync_init_struct.sync_polarity = DMAMUX_SYNC_POLARITY_RISING;
edmamux_sync_init_struct.sync_event_enable = TRUE;
edmamux_sync_init_struct.sync_enable = TRUE;
edmamux_sync_config(EDMAMUX_CHANNEL4, &edmamux_sync_init_struct);
```

5.10.25 edmamux_generator_default_para_init function

The table below describes the function edmamux_generator_default_para_init.

Table 262. edmamux_generator_default_para_init function

Name	Description
Function name	edmamux_generator_default_para_init
Function prototype	void edmamux_generator_default_para_init(edmamux_gen_init_type * edmamux_gen_init_struct);
Function description	Initialize parameters in the edmamux_gen_init_struct
Input parameter 1	edmamux_gen_init_struct: edmamux_gen_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of the edmamux_gen_init_struct members.

Table 263.edmamux_gen_init_struct default values

Member	Default value
gen_signal_sel	(edmamux_gen_id_sel_type)0x0
gen_polarity	EDMAMUX_GEN_POLARITY_DISABLE
gen_request_number	0x0
gen_enable	FALSE

Example:

```
/* edmamux gen init config with its default value */
edmamux_gen_init_type edmamux_gen_init_struct = {0};
edmamux_gen_default_para_init (&edmamux_gen_init_struct);
```

5.10.26 edmamux_generator_config function

The table below describes the function edmamux_generator_config.

Table 264. edmamux_generator_config function

Name	Description
Function name	edmamux_generator_config
Function prototype	void edmamux_generator_config(edmamux_generator_type * edmamux_gen_x, edmamux_gen_init_type *edmamux_gen_init_struct);
Function description	Configure EDMAMUX request generator
Input parameter 1	*xmc_sdram_init_***struct** : request generator channel
Input parameter 2	edmamux_sync_init_struct: edmamux_sync_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

edmamux_gen_x

This is used to select DMA request generator channel.

EDMAMUX_GENERATOR1

EDMAMUX_GENERATOR2

EDMAMUX_GENERATOR3

EDMAMUX_GENERATOR4

edmamux_sync_init_struct

The edmamux_gen_init_type is defined in the at32f435_437_dma.h.

```
typedef struct
{
    edmamux_gen_id_sel_type        gen_signal_sel;
    uint32_t                       gen_polarity;
    uint32_t                       gen_request_number;
    confirm_state                  gen_enable;
} edmamux_gen_init_type;
```

gen_signal_sel

Select signal source for synchronous module

EDMAMUX_GEN_ID_EXINT0: External exint0 signal

EDMAMUX_GEN _ID_EXINT1: External exint1 signal

EDMAMUX_GEN _ID_EXINT2: External exint2 signal

EDMAMUX_GEN _ID_EXINT3: External exint3 signal

EDMAMUX_GEN_ID_EXINT4: External exint4 signal

EDMAMUX_GEN_ID_EXINT5: External exint5 signal

EDMAMUX_GEN_ID_EXINT6: External exint6 signal

EDMAMUX_GEN_ID_EXINT7: External exint7 signal

EDMAMUX_GEN_ID_EXINT8: External exint8 signal

EDMAMUX_GEN_ID_EXINT9: External exint9 signal

EDMAMUX_GEN_ID_EXINT10: External exint10 signal

EDMAMUX_GEN_ID_EXINT11: External exint11 signal

EDMAMUX_GEN_ID_EXINT12: External exint12 signal

EDMAMUX_GEN_ID_EXINT13: External exint13 signal
EDMAMUX_GEN_ID_EXINT14: External exint14 signal
EDMAMUX_GEN_ID_EXINT15: External exint15 signal
EDMAMUX_GEN_ID_DMAMUX_CH1_EVT: dmamux channel 1 event
EDMAMUX_GEN_ID_DMAMUX_CH2_EVT: dmamux channel 2 event
EDMAMUX_GEN_ID_DMAMUX_CH3_EVT: dmamux channel 3 event
EDMAMUX_GEN_ID_DMAMUX_CH4_EVT: dmamux channel 4 event
EDMAMUX_GEN_ID_DMAMUX_CH5_EVT: dmamux channel 5 event
EDMAMUX_GEN_ID_DMAMUX_CH6_EVT: dmamux channel 6 event
EDMAMUX_GEN_ID_DMAMUX_CH7_EVT: dmamux channel 7 event

gen_polarity

Select generator signal polarity

EDMAMUX_GEN_POLARITY_RISING: Rising edge
EDMAMUX_GEN_POLARITY_FALLING: Falling edge
EDMAMUX_GEN_POLARITY_RISING_FALLING: Rising edge and falling edge

gen_request_number

Number of DMA request generated by the generator

Range: 1~32

gen_enable

Enable or disable request generator

FALSE: Disabled
TRUE: Enabled

Example:

```
/* genertor1 configuration */
edmamux_generator_default_para_init(&edmamux_gen_init_struct);
edmamux_gen_init_struct.gen_polarity = EDMAMUX_GEN_POLARITY_RISING;
edmamux_gen_init_struct.gen_request_number = 4;
edmamux_gen_init_struct.gen_signal_sel = EDMAMUX_GEN_ID_EXINT1;
edmamux_gen_init_struct.gen_enable = TRUE;
edmamux_generator_config(EDMAMUX_GENERATOR1, &edmamux_gen_init_struct);
```

5.10.27 edmamux_sync_interrupt_enable function

The table below describes the function edmamux_sync_interrupt_enable.

Table 265. edmamux_sync_interrupt_enable function

Name	Description
Function name	edmamux_sync_interrupt_enable
Function prototype	void edmamux_sync_interrupt_enable(edmamux_channel_type *edmamux_channelx, confirm_state new_state);
Function description	Enable synchronous module overrun interrupt
Input parameter 1	*edmamux_channelx* : EDMAMUX channel, x=1…7
Input parameter 2	new_state: enable or disable interrupt
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

Enable or disable the DMA channel interrupt.

FALSE: Interrupt disabled

TRUE: Interrupt enabled

Example:

```
/* enable sync overrun intterrupt */
edmamux_sync_interrupt_enable (EDMAMUX_CHANNEL1, TRUE);
```

5.10.28 edmamux_generator_interrupt_enable function

The table below describes the function edmamux_generator_interrupt_enable.

Table 266. edmamux_generator_interrupt_enable function

Name	Description
Function name	edmamux_generator_interrupt_enable
Function prototype	void edmamux_generator_interrupt_enable(edmamux_generator_type *edmamux_gen_x, confirm_state new_state);
Function description	Enable request generator overrun interrupt
Input parameter 1	*xmc_sdram_init_*__struct__ : EDMAMUX request generator channel, x=1…4
Input parameter 2	new_state: enable or disable interrupt
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

Enable or disable the request generator channel interrupt.

FALSE: Interrupt disabled

TRUE: Interrupt enabled

Example:

```
/* enable gen overrun intterrupt */
edmamux_generator_interrupt_enable(EDMAMUX_GENERATOR3, TRUE);
```

5.10.29 edmamux_sync_flag_get function

The table below describes the function edmamux_sync_flag_get.

Table 267. edmamux_sync_flag_get function

Name	Description
Function name	edmamux_sync_flag_get
Function prototype	flag_status edmamux_sync_flag_get(dma_type *dma_x, uint32_t flag);
Function description	Get the edmamux synchronous module flag
Input parameter 1	flag: the desired flag
Output parameter	NA
Return value	flag_status: indicates whether or not the flag is set
Required preconditions	NA
Called functions	NA

flag

The flag is used to select the desired status flag, including:

EDMAMUX_SYNC_OV1_FLAG
EDMAMUX_SYNC_OV2_FLAG
EDMAMUX_SYNC_OV3_FLAG
EDMAMUX_SYNC_OV4_FLAG
EDMAMUX_SYNC_OV5_FLAG
EDMAMUX_SYNC_OV6_FLAG
EDMAMUX_SYNC_OV7_FLAG
EDMAMUX_SYNC_OV8_FLAG

flag_status

RESET: Corresponding flag is reset

SET: Corresponding flag is set

Example:

```
if(edmamux_sync_flag_get (EDMAMUX_SYNC_OV1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
}
```

5.10.30 edmamux_sync_flag_clear function

The table below describes the function edmamux_sync_flag_clear.

Table 268. edmamux_sync_flag_clear function

Name	Description
Function name	edmamux_sync_flag_clear
Function prototype	void edmamux_sync_flag_clear(dma_type *dma_x, uint32_t flag);
Function description	Clear the synchronous module flag
Input parameter 1	flag: the flag to be cleared
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flag

The flag is used to select the flag to be cleared, including:

EDMAMUX_SYNC_OV1_FLAG
EDMAMUX_SYNC_OV2_FLAG
EDMAMUX_SYNC_OV3_FLAG
EDMAMUX_SYNC_OV4_FLAG
EDMAMUX_SYNC_OV5_FLAG
EDMAMUX_SYNC_OV6_FLAG
EDMAMUX_SYNC_OV7_FLAG
EDMAMUX_SYNC_OV8_FLAG

Example:

```
if(edmamux_sync_flag_get (EDMAMUX_SYNC_OV1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
    edmamux_sync_flag_clear( EDMAMUX_SYNC_OV1_FLAG);
}
```

5.10.31 edmamux_generator_flag_get function

The table below describes the function edmamux_generator_flag_get.

Table 269. edmamux_generator_flag_get function

Name	Description
Function name	edmamux_ generator_flag_get
Function prototype	flag_status edmamux_ generator_flag_get(dma_type *dma_x, uint32_t flag);
Function description	Get the edmamux request generator flag
Input parameter 1	flag: the desired flag
Output parameter	NA
Return value	flag_status: indicates whether or not the flag is set
Required preconditions	NA
Called functions	NA

flag

The flag is used to select the desired flag, including:

EDMAMUX_GEN_TRIG_OV1_FLAG

EDMAMUX_GEN_TRIG_OV2_FLAG

EDMAMUX_GEN_TRIG_OV3_FLAG

EDMAMUX_GEN_TRIG_OV4_FLAG

flag_status

RESET: Corresponding flag is reset

SET: Corresponding flag is set

Example:

```
if(edmamux_generator_flag_get (DMA2, EDMAMUX_GEN_TRIG_OV1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
}
```

5.10.32 edmamux_generator_flag_clear function

The table below describes the function edmamux_generator_flag_clear.

Table 270. edmamux_generator_flag_clear function

Name	Description
Function name	edmamux_generator_flag_clear
Function prototype	void edmamux_generator_flag_clear(dma_type *dma_x, uint32_t flag);
Function description	Clear the request generator flag
Input parameter 1	flag: the flag to be cleared
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flag

The flag is used to select the flag to be cleared, including:

EDMAMUX_GEN_TRIG_OV1_FLAG

EDMAMUX_GEN_TRIG_OV2_FLAG

EDMAMUX_GEN_TRIG_OV3_FLAG

EDMAMUX_GEN_TRIG_OV4_FLAG

Example:

```
if(edmamux_generator_flag_get (DMA2, EDMAMUX_GEN_TRIG_OV1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
    edmamux_generator_flag_clear(EDMAMUX_GEN_TRIG_OV1_FLAG);
}
```

5.11 Real-time clock (ERTC)

The ERTC register structure ertc_type is defined in the “at32f435_437_ertc.h”.

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


} ertc_type;
```

The table below gives a list of the ERTC registers.

Table 271. 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
ccal	ERTC coarse calibration register
ala	ERTC alarm clock A register
alb	ERTC alarm clock B 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
albsbs	ERTC alarm clock B subsecond register
bprx	ERTC battery powered domain data register

The table below gives a list of ERTC library functions.

Table 272. 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_coarse_calibration_set	Configure coarse calibration
ertc_coarse_calibration_enable	Enable coarse 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_pin_select	Time stamp detection pin selection
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_1_pin_select	Tamper detection 1 pin selection
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 data from battery powered data register (BPR)

5.11.1 ertc_num_to_bcd function

The table below describes the function ertc_num_to_bcd.

Table 273. 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);
```

5.11.2 ertc_bcd_to_num function

The table below describes the function ertc_bcd_to_num.

Table 274. 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);
```

5.11.3 ertc_write_protect_enable function

The table below describes the function ertc_write_protect_enable.

Table 275. 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();
```

5.11.4 ertc_write_protect_disable function

The table below describes the function ertc_write_protect_disable.

Table 276. 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();
```

5.11.5 ertc_wait_update function

The table below describes the function ertc_wait_update.

Table 277. 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 ERROR: flag wait timeout
Required preconditions	NA
Called functions	NA

Example:

```
ertc_wait_update();
```

5.11.6 ertc_wait_flag function

The table below describes the function ertc_wait_flag.

Table 278. 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 Refer to the “flag” description below for details.
Input parameter 1	status: flag status After the flag status is set, the function remains stuck here until flag status changes. This parameter can be SET or RESET.
Output parameter	NA
Return value	SUCCESS: flag status changed ERROR: flag wait timeout
Required preconditions	NA
Called functions	NA

flag

Flag selection

ERTC_ALAWF_FLAG: Alarm A write enable flag

ERTC_ALBWF_FLAG: Alarm B 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);
```

5.11.7 ertc_init_mode_enter function

The table below describes the function ertc_init_mode_enter.

Table 279. 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 ERROR: Timeout
Required preconditions	NA
Called functions	NA

Example:

```
ertc_init_mode_enter();
```

5.11.8 ertc_init_mode_exit function

The table below describes the function ertc_init_mode_exit.

Table 280. 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();
```

5.11.9 ertc_reset function

The table below describes the function ertc_reset.

Table 281. 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 success ERROR: reset failure
Required preconditions	NA
Called functions	NA

Example:

```
ertc_reset();
```

5.11.10 ertc_divider_set function

The table below describes the function ertc_divider_set.

Table 282. 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 settiings, frequency division value (div_a + 1) * (div_b + 1) = ERTC_CLK frequency For example, if 32768 Hz is used, the frequency division should be diva = 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 ERROR: Reset failed
Required preconditions	NA
Called functions	NA

Example:

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

5.11.11 ertc_hour_mode_set function

The table below describes the function ertc_hour_mode_set.

Table 283. 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 Refer to the following description “Mode” for details.
Output parameter	NA
Return value	SUCCESS: Setting success 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);
```

5.11.12 ertc_date_set function

The table below describes the function ertc_date_set.

Table 284. 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: 0~99
Input parameter 2	month: 1~12
Input parameter 3	date: 1~31
Input parameter 4	weekday: 1~7
Output parameter	NA
Return value	SUCCESS: Setting success ERROR: Setting error
Required preconditions	NA
Called functions	NA

Example:

```
ertc_date_set(22, 5, 26, 4);
```

5.11.13 ertc_time_set function

The table below describes the function ertc_time_set.

Table 285. 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: 0~23
Input parameter 2	min: 0~59
Input parameter 3	sec: 0~59
Input parameter 4	ampm: 12 AM/PM in 12-hour format (for 12-hour format only, don't care in 24-hour format) Refer to the following description "ampm" for details.
Output parameter	NA
Return value	SUCCESS: Setting success 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);
```

5.11.14 ertc_calendar_get function

The table below describes the function ertc_calendar_get.

Table 286. 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 at32f435_437_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, don’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);
```

5.11.15 ertc_sub_second_get function

The table below describes the function ertc_sub_second_get.

Table 287. 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();
```

5.11.16 ertc_alarm_mask_set function

The table below describes the function ertc_alarm_mask_set.

Table 288. 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 Refer to the following description “alarm_x” for details.
Input parameter 1	mask: Set alarm mask 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

ERTC_ALB: Alarm B

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);
```

5.11.17 ertc_alarm_week_date_select function

The table below describes the function ertc_alarm_week_date_select.

Table 289. 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, ertc_week_date_select_type wk);
Function description	Alarm time format selection: week/date
Input parameter 1	alarm_x: alarm selection Refer to the following description “alarm_x” for details.
Input parameter 2	wk: alarm week/date format selection 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

ERTC_ALB: Alarm B

wk

Alarm week/date format selection

ERTC_SLECT_DATE: Date mode

ERTC_SLECT_WEEK: Week mode

Example:

```
ertc_alarm_week_date_select(ERTC_ALA, ERTC_SLECT_DATE);
```

5.11.18 ertc_alarm_set function

The table below describes the function ertc_alarm_set.

Table 290. 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 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() Date: 1~31 Weekday: 1~7
Input parameter 3	hour: 0~23
Input parameter 4	min: 0~59
Input parameter 5	sec: 0~59
Input parameter 6	ampm: AM/PM in 12-hour format (12 hour format only, don’t care in 24-hour format) 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

ERTC_ALB: Alarm B

ampm

AM/PM in 12-hour format (for 12 hour format only, don’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);
```

5.11.19 ertc_alarm_sub_second_set function

The table below describes the function ertc_alarm_sub_second_set.

Table 291. 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 Refer to the following description “alarm_x” for details.
Input parameter 2	value: subsecond value, range 0~0x7FFF
Input parameter 3	mask: alarm mask settings 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

ERTC_ALB: Alarm B

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);
```

5.11.20 ertc_alarm_enable function

The table below describes the function ertc_alarm_enable.

Table 292. 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 Refer to the following description “alarm_x” for details.
Input parameter 2	new_state: alarm enable status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	SUCCESS: Setting success ERROR: Setting error
Required preconditions	NA
Called functions	NA

alarm_x

Alarm selection

ERTC_ALA: Alarm A

ERTC_ALB: Alarm B

Example:

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

5.11.21 ertc_alarm_get function

The table below describes the function ertc_alarm_get.

Table 293. 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 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_ALB: Alarm B

ertc_alarm_value_type* alarm

The ertc_alarm_value_type is defined in the at32f435_437_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, don'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, generate an alarm per one second

week_date_sel

Alarm week/date format, including

ERTC_SLECT_DATE:	date mode
ERTC_SLECT_WEEK:	week mode

week

Range: 1~7

Example:

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

5.11.22 ertc_alarm_sub_second_get function

The table below describes the function ertc_alarm_sub_second_get.

Table 294. 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 Refer to the following description “alarm_x” for details.
Output parameter	NA
Return value	Alarm subsecond
Required preconditions	NA
Called functions	NA

alarm_x

Alarm selection

ERTC_ALA: Alarm A

ERTC_ALB: Alarm B

Example:

```
ertc_alarm_sub_second_get(ERTC_ALA);
```

5.11.23 ertc_wakeup_clock_set function

The table below describes the function ertc_wakeup_clock_set.

Table 295. 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 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);
```

5.11.24 ertc_wakeup_counter_set function

The table below describes the function ertc_wakeup_counter_set.

Table 296. 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: counter value, range: 0~0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.11.25 ertc_wakeup_counter_get function

The table below describes the function ertc_wakeup_counter_get.

Table 297. 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();
```

5.11.26 ertc_wakeup_enable function

The table below describes the function ertc_wakeup_enable.

Table 298. 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 This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	SUCCESS: Setting success ERROR: Setting error
Required preconditions	NA
Called functions	NA

Example:

```
ertc_wakeup_enable(TRUE);
```

5.11.27 ertc_smooth_calibration_config function

The table below describes the function ertc_smooth_calibration_config.

Table 299. 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	Set smooth digital calibration
Input parameter 1	period: calibration period Refer to the following “period” descriptions for details.
Input parameter 2	clk_add: add ERTC CLK cycles Refer to the following “clk_add” descriptions for details.
Input parameter 3	clk_dec: reduce ERTC CLK cycles, range 0~511
Output parameter	NA
Return value	SUCCESS: Setting success ERROR: Setting error
Required preconditions	NA
Called functions	NA

period

Calibration period

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);
```

5.11.28 ertc_coarse_calibration_set function

The table below describes the function ertc_coarse_calibration_set.

Table 300. ertc_coarse_calibration_set function

Name	Description
Function name	ertc_coarse_calibration_set
Function prototype	error_status ertc_coarse_calibration_set(ertc_cal_direction_type dir, uint32_t value);
Function description	Set coarse digital calibration
Input parameter 1	dir: calibration direction Refer to the following “dir” descriptions for details.
Input parameter 2	value: calibration value, range 0~31
Output parameter	NA
Return value	SUCCESS: Setting success ERROR: Setting error
Required preconditions	NA
Called functions	NA

dir

Calibration direction

ERTC_CAL_DIR_POSITIVE: Positive directin calibration

ERTC_CAL_DIR_NEGATIVE: Negative directin calibration

Example:

```
ertc_coarse_calibration_set(ERTC_CAL_DIR_POSITIVE, 10);
```

5.11.29 ertc_coarse_calibration_enable function

The table below describes the function ertc_coarse_calibration_enable.

Table 301. ertc_coarse_calibration_enable function

Name	Description
Function name	ertc_coarse_calibration_enable
Function prototype	error_status ertc_coarse_calibration_enable(confirm_state new_state);
Function description	Enable coarse digital calibration
Input parameter 1	new_state: coarse digital calibration enable state This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	SUCCESS: Setting success ERROR: Setting error
Required preconditions	NA
Called functions	NA

Example:

```
ertc_coarse_calibration_enable(TRUE);
```

5.11.30 ertc_cal_output_select function

The table below describes the function ertc_cal_output_select.

Table 302. 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 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);
```

5.11.31 ertc_cal_output_enable function

The table below describes the function ertc_cal_output_enable.

Table 303. 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 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);
```

5.11.32 ertc_time_adjust function

The table below describes the function ertc_time_adjust.

Table 304. 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 Refer to the following “add1s” descriptions for details.
Input parameter 2	decsbs: reduce subseconds, range 0~0x7FFF
Output parameter	NA
Return value	SUCCESS: setting success ERROR: setting 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);
```

5.11.33 ertc_daylight_set function

The table below describes the function ertc_daylight_set.

Table 305. 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 Refer to the following “operation” descriptions for details.
Input parameter 2	save: save daylight time 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=0 in the CTRL register

ERTC_DST_SAVE_1: Set BPR=1 in the CTRL register

Example:

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

5.11.34 ertc_daylight_bpr_get function

The table below describes the function ertc_daylight_bpr_get.

Table 306. 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();
```

5.11.35 ertc_refer_clock_detect_enable function

The table below describes the function ertc_refer_clock_detect_enable.

Table 307. 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 This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	SUCCESS: Setting success ERROR: Setting error
Required preconditions	NA
Called functions	NA

Example:

```
ertc_refer_clock_detect_enable(TRUE);
```

5.11.36 ertc_direct_read_enable function

The table below describes the function ertc_direct_read_enable

Table 308. 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 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);
```

5.11.37 ertc_output_set function

The table below describes the function ertc_output_set.

Table 309. 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 Refer to the following “source” descriptions for details.
Input parameter 2	polarity: output polarity Refer to the following “polarity” descriptions for details.
Input parameter 3	type: output type 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);
```

5.11.38 ertc_timestamp_pin_select function

The table below describes the function ertc_timestamp_pin_select.

Table 310. ertc_timestamp_pin_select function

Name	Description
Function name	ertc_timestamp_pin_select
Function prototype	void ertc_timestamp_pin_select(ertc_pin_select_type pin);
Function description	Timestamp detection pin selection
Input parameter 1	pin: Timestamp detection pin Refer to the following “pin” descriptions for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pin

Timestampe detection pin

ERTC_PIN_PC13: PC13 is selected as timestampe detection pin

ERTC_PIN_PA0: PA0 is selected as timestampe detection pin

Example:

```
ertc_timestamp_pin_select(ERTC_PIN_PC13);
```

5.11.39 ertc_timestamp_valid_edge_set function

The table below describes the function ertc_timestamp_valid_edge_set.

Table 311. 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 timestampe detection valid edge
Input parameter 1	edge: timestampe detection valid edge Refer to the following “edge” descriptions for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

edge

Timestampe detection valid edge

ERTC_TIMESTAMP_EDGE_RISING: Rising edge

ERTC_TIMESTAMP_EDGE_FALLING: Falling edge

Example:

```
ertc_timestamp_valid_edge_set(ERTC_TIMESTAMP_EDGE_RISING);
```

5.11.40 ertc_timestamp_enable function

The table below describes the function ertc_timestamp_enable.

Table 312. 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 This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
ertc_timestamp_enable(TRUE);
```

5.11.41 ertc_timestamp_get function

The table below describes the function ertc_timestamp_get.

Table 313. 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 at32f435_437_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, don'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);
```

5.11.42 ertc_timestamp_sub_second_get function

The table below describes the function ertc_timestamp_sub_second_get.

Table 314. 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();
```

5.11.43 ertc_tamper_1_pin_select function

The table below describes the function ertc_tamper_1_pin_select.

Table 315. ertc_tamper_1_pin_select function

Name	Description
Function name	ertc_tamper_1_pin_select
Function prototype	void ertc_tamper_1_pin_select(ertc_pin_select_type pin);
Function description	Tamper detection 1 pin selection
Input parameter 1	pin: Tamper detection pin Refer to the following “pin” descriptions for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pin

Tamper detection pin

ERTC_PIN_PC13: PC13 is selected as a tamper detection pin

ERTC_PIN_PA0: PA0 is selected as a tamper detection pin

Example:

```
ertc_tamper_1_pin_select(ERTC_PIN_PC13);
```

5.11.44 ertc_tamper_pull_up_enable function

The table below describes the function ertc_tamper_pull_up_enable.

Table 316. 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 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);
```

5.11.45 ertc_tamper_precharge_set function

The table below describes the function ertc_tamper_precharge_set.

Table 317. ertc_tamper_precharge_set function

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

precharge

Tamper pin precharge 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_precharge_set(ERTC_TAMPER_PR_2_ERTCCLK);
```

5.11.46 ertc_tamper_filter_set function

The table below describes the function ertc_tamper_filter_set.

Table 318. 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 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 samplings

ERTC_TAMPER_FILTER_4:

Tamper event is considered to have occur after four valid consecutive samplings

ERTC_TAMPER_FILTER_8:

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

Example:

```
ertc_tamper_filter_set(ERTC_TAMPER_FILTER_2);
```

5.11.47 ertc_tamper_detect_freq_set function

The table below describes the function ertc_tamper_detect_freq_set.

Table 319. 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 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);
```

5.11.48 ertc_tamper_valid_edge_set function

The table below describes the function ertc_tamper_valid_edge_set.

Table 320. 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, ertc_tamper_valid_edge_type trigger);
Function description	Set tamper detection valid edge
Input parameter 1	tamper_x: tamper selection Refer to the following “tamper_x” descriptions for details.
Input parameter 2	trigger: tamper detection valid edge 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 selection 1
ERTC_TAMPER_2: Tamper selection 2

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);
```

5.11.49 ertc_tamper_timestamp_enable function

The table below describes the function ertc_tamper_timestamp_enable.

Table 321. 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 occurs
Input parameter 1	new_state: timestamp feature enable state when a tamper event occurs 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);
```

5.11.50 ertc_tamper_enable function

The table below describes the function ertc_tamper_enable.

Table 322. 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 detection Refer to the following “tamper_x” descriptions for details.
Input parameter 2	new_state: tamper detection enable state 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 selection 1

ERTC_TAMPER_2: Tamper selection 2

Example:

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

5.11.51 ertc_interrupt_enable function

The table below describes the function ertc_interrupt_enable.

Table 323. 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 Refer to the following “source” descriptions for details.
Input parameter 1	new_state: interrupt enable state 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_ALB_INT: Alarm B interrupt

ERTC_WAT_INT: Wakeup timer interrupt

ERTC_TS_INT: Time stamp interrupt

Example:

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

5.11.52 ertc_interrupt_get function

The table below describes the function ertc_interrupt_get.

Table 324. 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 enable state Refer to the following “source” descriptions for details.
Output parameter	NA
Return value	flag_status: flag status 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_ALB_INT: Alarm B interrupt
ERTC_WAT_INT: Wakeup timer interrupt
ERTC_TS_INT: Time stamp interrupt

Example:

```
ertc_interrupt_get(ERTC_TP_INT);
```

5.11.53 ertc_flag_get function

The table below describes the function ertc_flag_get.

Table 325. 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 Refer to the following “flag” descriptions for details.
Output parameter	NA
Return value	flag_status: flag status This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

flag

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

ERTC_ALAWF_FLAG: Alarm A write enable flag
ERTC_ALBWF_FLAG: Alarm B 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_ALBF_FLAG: Alarm B flag
ERTC_WATF_FLAG: Wakeup timer flag
ERTC_TSF_FLAG: Time stamp flag
ERTC_TSOF_FLAG: Time stamp overflow flag
ERTC_TP1F_FLAG: Tamper detection 1 flag
ERTC_TP2F_FLAG: Tamper detection 2 flag
ERTC_CALUPDF_FLAG: Calibration value update complete flag

Example:

```
ertc_flag_get(ERTC_TP1F_FLAG);
```

5.11.54 ertc_interrupt_flag_get function

The table below describes the function ertc_interrupt_flag_get.

Table 326. 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 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_ALBF_FLAG: Alarm B flag
ERTC_WATF_FLAG: Wakeup timer flag
ERTC_TSF_FLAG: Time stamp flag
ERTC_TSOF_FLAG: Time stamp overflow flag
ERTC_TP1F_FLAG: Tamper detection 1 flag
ERTC_TP2F_FLAG: Tamper detection 2 flag

Example:

```
ertc_interrupt_flag_get(ERTC_TP1F_FLAG);
```

5.11.55 ertc_flag_clear function

The table below describes the function ertc_flag_clear.

Table 327. ertc_flag_clear function

Name	Description
Function name	ertc_flag_clear
Function prototype	void ertc_flag_clear(uint32_t flag);
Function description	Clear flag
Input parameter 1	flag: flag selection 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_ALBWF_FLAG: Alarm B 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_ALBF_FLAG: Alarm B flag
ERTC_WATF_FLAG: Wakeup timer flag
ERTC_TSF_FLAG: Time stamp flag
ERTC_TSOF_FLAG: Time stamp overflow flag
ERTC_TP1F_FLAG: Tamper detection 1 flag
ERTC_TP2F_FLAG: Tamper detection 2 flag
ERTC_CALUPDF_FLAG: Calibration value update complete flag

Example:

```
ertc_flag_clear(ERTC_TP1F_FLAG);
```

5.11.56 ertc_bpr_data_write function

The table below describes the function ertc_bpr_data_write.

Table 328. 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 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_DT19: Data register 19
ERTC_DT20: Data register 20

Example:

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

5.11.57 ertc_bpr_data_read function

The table below describes the function ertc_bpr_data_read.

Table 329. 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 Refer to the following “dt” descriptions for details.
Output parameter	NA
Return value	BPR register data
Required preconditions	NA
Called functions	NA

dt

Data register

ERTC_DT1: Data register 1

ERTC_DT2: Data register 2

...

ERTC_DT19: Data register 19

ERTC_DT20: Data register 20

Example:

```
ertc_bpr_data_read(ERTC_DT1);
```

5.12 External interrupt/event controller (EXINT)

The EXINT register structure exint_type is defined in the “at32f435_437_exint.h”.

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

The table below gives a list of the EXINT registers:

Table 330. 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 331. Summary of EXINT library functions

Function name	Description
exint_reset	Reset all EXINT registers to their reset values
exint_default_para_init	Configure the EXINT initial structure with the initial value
exint_init	Initialize EXINT
exint_flag_clear	Clear the selected EXINT interrupt flag
exint_flag_get	Read the selected EXINT interrupt flag
exint_interrupt_flag_get	Get 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

5.12.1 exint_reset function

The table below describes the function exint_reset.

Table 332. 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

Name	Description
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

Example:

```
exint_reset ();
```

5.12.2 exint_default_para_init function

The table below describes the function exint_default_para_init.

Table 333. 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 initial value
Input parameter 1	exint_struct: *exint_init_type* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of the exint_init_type before starting.
Called functions	NA

Example:

```
exint_init_type exint_init_struct;
exint_default_para_init(&exint_init_struct);
```

5.12.3 exint_init function

The table below describes the function exint_init.

Table 334. 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	*exint_init_type:* exint_init_struct pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of the exint_init_type before starting.
Called functions	NA

The exint_init_type is defined in the at32f435_437_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_INTERRUPUT: Interrupt mode
EXINT_LINE_EVENT: Event mode
line_select
Line selection
EXINT_LINE_NONE: No line
EXINT_LINE_0: line0
EXINT_LINE_1: line1
…
EXINT_LINE_21: line21
EXINT_LINE_22: line22
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_INTERRUPUT;
exint_init_struct.line_select = EXINT_LINE_0;
exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;
exint_init(&exint_init_struct);
```

5.12.4 exint_flag_clear function

The table below describes the function exint_flag_clear.

Table 335. exint_flag_clear function

Name	Description
Function name	exint_flag_clear
Function prototype	void exint_flag_clear(uint32_t exint_line);
Function description	Clear the selected EXINT interrupt flag
Input parameter	exint_line: line selection Refer to the *line_select* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
exint_flag_clear(EXINT_LINE_0);
```

5.12.5 exint_flag_get function

The table below describes the function exint_flag_get.

Table 336. exint_flag_get function

Name	Description
Function name	exint_flag_get
Function prototype	flag_status exint_flag_get(uint32_t exint_line);
Function description	Get the selected EXINT interrupt flag
Input parameter	exint_line: line selection Refer to the *line_select* for details.
Output parameter	NA
Return value	flag_status: indicates the status of the selected flag This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

```
flag_status status = RESET;
status = exint_flag_get(EXINT_LINE_0);
```

5.12.6 exint_interrupt_flag_get function

The table below describes the function exint_interrupt_flag_get.

Table 337. 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 Refer to the *line_select* 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);
```

5.12.7 exint_software_interrupt_event_generate function

The table below describes the function exint_software_interrupt_event_generate.

Table 338. exint_software_interrupt_event_generate function

Name	Description
Function name	exint_software_interrupt_event_generate
Function prototype	void exint_software_interrupt_event_generate(uint32_t exint_line);
Function description	Generate software interrupt event
Input parameter	exint_line: line selection Refer to the *line_select* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
exint_software_interrupt_event_generate (EXINT_LINE_0);
```

5.12.8 exint_interrupt_enable function

The table below describes the function exint_interrupt_enable.

Table 339. exint_interrupt_enable function

Name	Description
Function name	exint_interrupt_enable
Function prototype	void exint_interrupt_enable(uint32_t exint_line, confirm_state new_state);
Function description	Enable the selected EXINT interrupt
Input parameter 1	exint_line: line selection Refer to the *line_select* for details.
Input parameter 2	new_state: enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
exint_interrupt_enable (EXINT_LINE_0);
```

5.12.9 exint_event_enable function

The table below describes the function exint_event_enable.

Table 340. exint_event_enable function

Name	Description
Function name	exint_event_enable
Function prototype	void exint_event_enable(uint32_t exint_line, confirm_state new_state);
Function description	Enable the selected EXINT event
Input parameter 1	exint_line: line selection Refer to the *line_select* for details.
Input parameter 2	new_state: enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
exint_event_enable (EXINT_LINE_0);
```

5.13 Flash memory controller (FLASH)

The FLASH register structure flash_type is defined in the “at32f435_437_flash.h”.

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

The table below gives a list of the FLASH registers.

Table 341. 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_epps0	Erase/program protection status register 0
flash_epps1	Erase/program protection status register 1
flash_unlock2	Flash unlock register 2
flash_sts2	Flash status register 2
flash_ctrl2	Flash control register 2

Register	Description
flash_addr2	Flash address register 2
flash_contr	Flash continue read register
flash_divr	Flash divider register
slib_sts2	Flash security library status register 2
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	Security library additional status register
slib_set_pwd	Security library password setting register
slib_set_range0	Security library password setting register 0
slib_set_range1	Security library password setting register 1
slib_unlock	Security library unlock register
flash_crc_ctrl	Flash CRC calibration control register
flash_crc_chkr	Flash CRC check result register

The table below gives a list of FLASH library functions.

Table 342. 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_bank1_operation_status_get	Get Flash operation status (Flash memory bank 1)
flash_bank2_operation_status_get	Get Flash operation status (Flash memory bank 2)
flash_operation_wait_for	Wait for Flash operation complete
flash_bank1_operation_wait_for	Wait for Flash operation complete (Flash memory bank 1)
flash_bank2_operation_wait_for	Wait for Flash operation complete (Flash memory bank 2)
flash_unlock	Unlock Flash (Flash memory bank 1 and bank 2)
flash_bank1_unlock	Unlock Flash (Flash memory bank 1)
flash_bank2_unlock	Unlock Flash (Flash memory bank 2)
flash_lock	Lock Flash (Flash memory bank 1 and bank 2)
flash_bank1_lock	Lock Flash (Flash memory bank 1)
flash_bank2_lock	Lock Flash (Flash memory bank 2)
flash_sector_erase	Erase Flash sector
flash_block_erase	Erase Flash memory block
flash_internal_all_erase	Erase internal Flash
flash_bank1_erase	Erase Flash memory bank 1
flash_bank2_erase	Erase Flash memory bank 2
flash_user_system_data_erase	Erase user system data
flash_eopb0_config	Extended system option configuration
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

Function name	Description
flash_epp_status_get	Get erase/programming protection status
flash_fap_enable	Configure Flash access protection
flash_fap_status_get	Get Flash access protection status
flash_ssb_set	System configuration byte configuration
flash_ssb_status_get	Get system configuration byte configuration status
flash_interrupt_enable	Flash interrupt configuration
flash_slib_enable	sLib enable
flash_slib_disable	sLib disable
flash_slib_remaining_count_get	Get sLib remaining count
flash_slib_state_get	Get sLib status
flash_slib_start_sector_get	Get sLib start sector
flash_slib_inststart_sector_get	Get sLib instruction start sector
flash_slib_end_sector_get	Get sLib end sector
flash_crc_calibrate	Flash CRC verify
flash_nzw_boost_enable	Flash non-zero-wait area boost enable
flash_continue_read_enable	Flash continuous read enable

5.13.1 flash_flag_get function

The table below describes the function flash_flag_get.

Table 343. flash_flag_get function

Name	Description
Function name	flash_flag_get
Function prototype	flag_status flash_flag_get(uint32_t flash_flag);
Function description	Get flag status
Input parameter	flash_flag: flag selection
Output parameter	NA
Return value	flag_status: indicates the flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

flash_flag

Flag selection

FLASH_OBF_FLAG	Flash operation busy
FLASH_ODF_FLAG	Flash operation complete
FLASH_PRGMERR_FLAG	Flash programming error
FLASH_EPPERR_FLAG	Flash erase error
FLASH_BANK1_OBF_FLAG	Flash operation busy (bank 1)
FLASH_BANK1_ODF_FLAG	Flash operation complete (bank 1)
FLASH_BANK1_PRGMERR_FLAG	Flash programming error (bank 1)
FLASH_BANK1_EPPERR_FLAG	Flash erase error (bank 1)
FLASH_BANK2_OBF_FLAG	Flash operation busy (bank 2)
FLASH_BANK2_ODF_FLAG	Flash operation complete (bank 2)
FLASH_BANK2_PRGMERR_FLAG	Flash programming error (bank 2)

FLASH_BANK2_EPPERR_FLAG Flash erase error (bank 2)
FLASH_USDERR_FLAG User system data area error

Example:

```
flag_status status;
status = flash_flag_get (FLASH_ODF_FLAG);
```

5.13.2 flash_flag_clear function

The table below describes the function flash_flag_clear.

Table 344. flash_flag_clear function

Name	Description
Function name	flash_flag_clear
Function prototype	void flash_flag_clear(uint32_t flash_flag);
Function description	Clear flag
Input parameter	flash_flag: flag selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flash_flag

Flag selection

FLASH_ODF_FLAG Flash operation complete
FLASH_PRGMERR_FLAG Flash programming error
FLASH_EPPERR_FLAG Flash erase error
FLASH_BANK1_ODF_FLAG Flash operation complete (Flash bank 1)
FLASH_BANK1_PRGMERR_FLAG Flash programming error (Flash bank 1)
FLASH_BANK1_EPPERR_FLAG Flash erase error (Flash bank 1)
FLASH_BANK2_ODF_FLAG Flash operation complete (Flash bank 2)
FLASH_BANK2_PRGMERR_FLAG Flash programming error (Flash bank 2)
FLASH_BANK2_EPPERR_FLAG Flash erase error (Flash bank 2)

Example:

```
flash_flag_clear(FLASH_ODF_FLAG);
```

5.13.3 flash_operation_status_get function

The table below describes the function flash_operation_status_get.

Table 345. 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	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA

Name	Description
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	Flash operate timeout

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
/* check for the flash status */
status = flash_operation_status_get();
```

5.13.4 flash_bank1_operation_status_get function

The table below describes the function flash_bank1_operation_status_get.

Table 346. flash_bank1_operation_status_get function

Name	Description
Function name	flash_bank1_operation_status_get
Function prototype	flash_status_type flash_bank1_operation_status_get (void);
Function description	Get the Flash bank 1 operation status
Input parameter	NA
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
/* check for the flash status */
status = flash_bank1_operation_status_get();
```

5.13.5 flash_bank2_operation_status_get function

The table below describes the function flash_bank2_operation_status_get.

Table 347. flash_bank2_operation_status_get function

Name	Description
Function name	flash_bank2_operation_status_get
Function prototype	flash_status_type flash_bank2_operation_status_get (void);
Function description	Get the Flash bank 2 operation status
Input parameter	NA
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
/* check for the flash status */
status = flash_bank2_operation_status_get();
```

5.13.6 flash_operation_wait_for function

The table below describes the function flash_operation_wait_for.

Table 348. flash_operation_wait_for function

Name	Description
Function name	flash_operation_wait_for
Function prototype	flash_status_type flash_operation_wait_for(uint32_t time_out);
Function description	Wait for Flash operation
Input parameter	time_out: wait timeout The timeout value is defined in the flash.h file. Refer to *flash_time_out* for details.
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

flash_time_out

ERASE_TIMEOUT	Erase timeout
PROGRAMMING_TIMEOUT	Programming timeout
OPERATION_TIMEOUT	General operation timeout

Example:

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

5.13.7 flash_bank1_operation_wait_for function

The table below describes the function flash_bank1_operation_wait_for.

Table 349. flash_bank1_operation_wait_for function

Name	Description
Function name	flash_bank1_operation_wait_for
Function prototype	flash_status_type flash_bank1_operation_wait_for(uint32_t time_out);
Function description	Wait for Flash bank 1 operation
Input parameter	time_out: wait timeout The timeout value is defined in the flash.h file. Refer to *flash_time_out* for details.
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
/* wait for operation to be completed */
status = flash_bank1_operation_wait_for(PROGRAMMING_TIMEOUT);
```

5.13.8 flash_bank2_operation_wait_for function

The table below describes the function flash_bank2_operation_wait_for.

Table 350. flash_bank2_operation_wait_for function

Name	Description
Function name	flash_bank2_operation_wait_for
Function prototype	flash_status_type flash_bank2_operation_wait_for(uint32_t time_out);
Function description	Wait for Flash bank 2 operation
Input parameter	time_out: wait timeout The timeout value is defined in the flash.h file. Refer to *flash_time_out* for details.
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
/* wait for operation to be completed */
status = flash_bank2_operation_wait_for(PROGRAMMING_TIMEOUT);
```

5.13.9 flash_unlock function

The table below describes the function flash_unlock.

Table 351. flash_unlock function

Name	Description
Function name	flash_unlock
Function prototype	void flash_unlock(void);
Function description	Unlock Flash control register
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
flash_unlock();
```

5.13.10 flash_bank1_unlock function

The table below describes the function flash_bank1_unlock.

Table 352. flash_bank1_unlock function

Name	Description
Function name	flash_bank1_unlock
Function prototype	void flash_bank1_unlock(void);
Function description	Unlock Flash bank 1 control register
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
flash_bank1_unlock();
```

5.13.11 flash_bank2_unlock function

The table below describes the function flash_bank2_unlock.

Table 353. flash_bank2_unlock function

Name	Description
Function name	flash_bank2_unlock
Function prototype	void flash_bank2_unlock(void);
Function description	Unlock Flash bank 2 control register
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
flash_bank2_unlock();
```

5.13.12 flash_lock function

The table below describes the function flash_lock.

Table 354. flash_lock function

Name	Description
Function name	flash_lock
Function prototype	void flash_lock(void);
Function description	Lock Flash control register
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
flash_lock();
```

5.13.13 flash_bank1_lock function

The table below describes the function flash_bank1_lock.

Table 355. flash_bank1_lock function

Name	Description
Function name	flash_bank1_lock
Function prototype	void flash_bank1_lock(void);
Function description	Lock Flash bank 1 control register
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
flash_bank1_lock();
```

5.13.14 flash_bank2_lock function

The table below describes the function flash_bank2_lock.

Table 356. flash_bank2_lock function

Name	Description
Function name	flash_bank2_lock
Function prototype	void flash_bank2_lock(void);
Function description	Lock Flash bank 2 control register
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
flash_bank2_lock();
```

5.13.15 flash_sector_erase function

The table below describes the function flash_sector_erase.

Table 357. 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

Name	Description
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
status = flash_sector_erase(0x08001000);
```

5.13.16 flash_block_erase function

The table below describes the function flash_block_erase.

Table 358. flash_block_erase function

Name	Description
Function name	flash_block_erase
Function prototype	flash_status_type flash_block_erase(uint32_t block_address);
Function description	Erase data in the selected Flash block address
Input parameter	block_address: select the Flash block address to be erased, usually Flash block start address
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
status = flash_block_erase(0x08010000);
```

5.13.17 flash_internal_all_erase function

The table below describes the function flash_internal_all_erase.

Table 359. 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	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

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

5.13.18 flash_bank1_erase function

The table below describes the function flash_bank1_erase.

Table 360. flash_bank1_erase function

Name	Description
Function name	flash_bank1_erase
Function prototype	flash_status_type flash_bank1_erase(void);
Function description	Erase Flash bank 1 data
Input parameter	NA
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_bank1_unlock();
status = flash_bank1_erase();
```

5.13.19 flash_bank2_erase function

The table below describes the function flash_bank2_erase.

Table 361. flash_bank2_erase function

Name	Description
Function name	flash_bank2_erase
Function prototype	flash_status_type flash_bank2_erase(void);
Function description	Erase Flash bank 2 data
Input parameter	NA
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_bank2_unlock();
status = flash_bank2_erase();
```

5.13.20 flash_user_system_data_erase function

The table below describes the function flash_user_system_data_erase.

Table 362. flash_user_system_data_erase function

Name	Description
Function name	flash_user_system_data_erase
Function prototype	flash_status_type flash_user_system_data_erase(void);
Function description	Erase user system data
Input parameter	NA
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* 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();
```

5.13.21 flash_eopb0_config function

The table below describes the function flash_eopb0_config.

Table 363. flash_eopb0_config function

Name	Description
Function name	flash_eopb0_config
Function prototype	flash_status_type flash_eopb0_config(flash_usd_eopb0_type data);
Function description	Extended system option configuration
Input parameter	data: Extended system size; refer to flash_usd_eopb0_type.
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

flash_usd_eopb0_type

FLASH_EOPB0_SRAM_512K	On-chip 512 KB SRAM + 128 KB zero-wait-state Flash
FLASH_EOPB0_SRAM_448K	On-chip 448 KB SRAM + 192 KB zero-wait-state Flash
FLASH_EOPB0_SRAM_384K	On-chip 384 KB SRAM + 256 KB zero-wait-state Flash
FLASH_EOPB0_SRAM_320K	On-chip 320 KB SRAM + 320 KB zero-wait-state Flash
FLASH_EOPB0_SRAM_256K	On-chip 256 KB SRAM + 384 KB zero-wait-state Flash
FLASH_EOPB0_SRAM_192K	On-chip 192k KB SRAM + 448 KB zero-wait-state Flash
FLASH_EOPB0_SRAM_128K	On-chip 128k KB SRAM + 512 KB zero-wait-state Flash

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
status = flash_eopb0_config(FLASH_EOPB0_SRAM_512K);
```

5.13.22 flash_word_program function

The table below describes the function flash_word_program.

Table 364. flash_word_program function

Name	Description
Function name	flash_word_program
Function prototype	flash_status_type flash_word_program(uint32_t address, uint32_t data);
Function description	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	Operation status Refer to *flash_status_type* 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);
  }
}
```

5.13.23 flash_halfword_program function

The table below describes the function flash_halfword_program.

Table 365. 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	Operation status Refer to *flash_status_type* 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);
  }
}
```

5.13.24 flash_byte_program function

The table below describes the function flash_byte_program

Table 366. 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	Operation status Refer to *flash_status_type* 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);
  }
}
```

5.13.25 flash_user_system_data_program function

The table below describes the function flash_user_system_data_program.

Table 367. 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	Operation status Refer to *flash_status_type* for details.
Required preconditions	The programming operation can be allowed only when data and its inverse data in the user system data area are all 0xFF.
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
status = flash_user_system_data_erase();
if(status = FLASH_OPERATE_DONE)
{
  /* program user system data */
  status = flash_user_system_data_program(0x1FFFF804, 0x55);
}
```

5.13.26 flash_epp_set function

The table below describes the function flash_epp_set.

Table 368. flash_epp_set function

Name	Description
Function name	flash_epp_set
Function prototype	flash_status_type flash_epp_set(uint32_t *sector_bits);
Function description	Configure erase/programming protection
Input parameter	*sector_bits: Erase/programming protection sector address pointer. Each bit in bits [31:0] protects 4KB sectors, and each bit in bits [62:32] protects 128KB sectors. Setting this bit to 1 enables sector protection.
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* 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);
}
```

5.13.27 flash_epp_status_get function

The table below describes the function flash_epp_status_get.

Table 369. flash_epp_status_get function

Name	Description
Function name	flash_epp_status_get
Function prototype	void flash_epp_status_get(uint32_t *sector_bits);
Function description	Get the status of erase/programming protection
Input parameter	NA
Output parameter	*sector_bits: Erase/programming protection sector address pointer. Each bit in bits [31:0] protects 4KB sectors, and each bit in bits [62:32] protects 128KB sectors. 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);
```

5.13.28 flash_fap_enable function

The table below describes the function flash_fap_enable.

Table 370. flash_fap_enable function

Name	Description
Function name	flash_fap_enable
Function prototype	flash_status_type flash_fap_enable(confirm_state new_state);
Function description	Enable Flash access protection
Input parameter	new_state: Flash access protection status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* 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);
```

5.13.29 flash_fap_status_get function

The table below describes the function flash_fap_status_get.

Table 371. 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 Return SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

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

5.13.30 flash_ssb_set function

The table below describes the function flash_ssb_set.

Table 372. flash_ssb_set function

Name	Description
Function name	flash_ssb_set
Function prototype	flash_status_type flash_ssb_set(uint8_t usd_ssb);
Function description	Set the status of system setting bytes
Input parameter	usd_ssb: system setting bytes. Refer to *ssb_data_define* for details.
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

ssb_data_define

type 1:

USD_WDT_ATO_DISABLE Watchdog is disabled

USD_WDT_ATO_ENABLE Watchdog is enabled

type 2:

USD_DEPSLP_NO_RST No reset occurs when entering Deepsleep mode

USD_DEPSLP_RST Reset occurs when entering Deepsleep mode

type 3:USD_STDBY_NO_RST No reset occurs when entering Standby mode

USD_STDBY_RST Reset occurs when entering Standby mode

type 4:

FLASH_BOOT_FROM_BANK1 Start from Flash memory bank 1

FLASH_BOOT_FROM_BANK2 Start from Flash memory bank 2

type 5:

USD_WDT_DEPSLP_CONTINUE

WDT does not stop counting while entering Deepsleep mode

USD_WDT_DEPSLP_STOP

WDT stops counting while entering Deepsleep mode

type 6:

USD_WDT_STDBY_CONTINUE

WDT does not stop counting while entering Standby mode

USD_WDT_STDBY_STOP

WDT stops counting while entering 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);
  }
```

5.13.31 flash_ssb_status_get function

The table below describes the function flash_ssb_status_get.

Table 373. 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	Dystem setting bytes Refer to *ssb_data_define* for details.
Required preconditions	NA
Called functions	NA

Example:

```
uint8_t ssb_val;
ssb_val = flash_ssb_status_get();
```

5.13.32 flash_interrupt_enable function

The table below describes the function flash_interrupt_enable.

Table 374. 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 *flash_interrupt_type* for details.
Input parameter 2	new_state: interrupt status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flash_interrupt_type

FLASH_ERR_INT	Flash error interrupt
FLASH_ODF_INT	Flash operation complete interrupt
FLASH_BANK1_ERR_INT	Flash bank 1 error interrupt
FLASH_BANK1_ODF_INT	Flash bank 1 operation complete interrupt
FLASH_BANK2_ERR_INT	Flash bank 2 error interrupt
FLASH_BANK2_ODF_INT	Flash bank 2 operation complete interrupt

Example:

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

5.13.33 flash_slib_enable function

The table below describes the function flash_slib_enable

Table 375. flash_slib_enable function

Name	Description
Function name	flash_slib_enable
Function prototype	flash_status_type flash_slib_enable(uint32_t pwd, uint16_t start_sector, uint16_t inst_start_sector, uint16_t end_sector);
Function description	Enable security library (sLib) and configure 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 instruction area start sector number
Input parameter 4	end_sector: sLib end sector number
Output parameter	NA
Return value	Operation status Refer to *flash_status_type* for details.
Required preconditions	NA
Called functions	NA

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
status = flash_slib_enable(0x12345678, 0x04, 0x05, 0x06);
```

5.13.34 flash_slib_disable function

The table below describes the function flash_slib_disable.

Table 376. flash_slib_disable function

Name	Description
Function name	flash_slib_disable
Function prototype	error_status flash_slib_disable(uint32_t pwd);
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 This parameter can be ERROR 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);
```

5.13.35 flash_slib_remaining_count_get function

The table below describes the function flash_slib_remaining_count_get.

Table 377. flash_slib_remaining_count_get function

Name	Description
Function name	flash_slib_remaining_count_get
Function prototype	uint32_t flash_slib_remaining_count_get(void);
Function description	Get the sLib remaining count
Input parameter	NA
Output parameter	NA
Return value	Return the sLib remaining count
Required preconditions	NA
Called functions	NA

Example:

```
uint32_t num;
num = flash_slib_remaining_count_get();
```

5.13.36 flash_slib_state_get function

The table below describes the function flash_slib_state_get.

Table 378. 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 This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

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

5.13.37 flash_slib_start_sector_get function

The table below describes the function flash_slib_start_sector_get.

Table 379. 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();
```

5.13.38 flash_slib_inststart_sector_get function

The table below describes the function flash_slib_inststart_sector_get.

Table 380. 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();
```

5.13.39 flash_slib_end_sector_get function

The table below describes the function flash_slib_end_sector_get.

Table 381. 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();
```

5.13.40 flash_crc_calibrate function

The table below describes the function flash_crc_calibrate.

Table 382. 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_sector: CRC check start address
Input parameter 2	sector_cnt: CRC check sector count
Output parameter	NA
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);
```

5.13.41 flash_nzw_boost_enable function

The table below describes the function flash_nzw_boost_enable.

Table 383. flash_nzw_boost_enable function

Name	Description
Function name	flash_nzw_boost_enable
Function prototype	void flash_nzw_boost_enable(confirm_state new_state);
Function description	Enable Flash non-zero-wait area boost
Input parameter	new_state: non-zero-wait area boost status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
flash_nzw_boost_enable(TRUE);
```

5.13.42 flash_continue_read_enable function

The table below describes the function flash_continue_read_enable.

Table 384. flash_continue_read_enable function

Name	Description
Function name	flash_continue_read_enable
Function prototype	void flash_continue_read_enable(confirm_state new_state);
Function description	Enable Flash continuous read
Input parameter	new_state: continuous read status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
flash_continue_read_enable(TRUE);
```

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

The GPIO register structure gpio_type is defined in the “at32f435_437_gpio.h”.

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


} gpio_type;
```

The table below gives a list of the GPIO registers.

Table 385. Summary of GPIO registers

Register	Description
cfgr	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 data register
odt	GPIO output data 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

The table below gives a list of the GPIO and IOMUX library functions.

Table 386. Summary of 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_toggle	Toggle GPIO
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

5.14.1 gpio_reset function

The table below describes the function gpio_reset.

Table 387. 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: Selected GPIO peripheral. This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG or GPIOH.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

Example:

```
gpio_reset(GPIOA);
```

5.14.2 gpio_init function

The table below describes the function gpio_init.

Table 388. 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: Selected GPIO peripheral. This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG or GPIOH.
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 at32f435_437_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);
```

5.14.3 gpio_default_para_init function

The table below describes the function gpio_default_para_init.

Table 389. 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 gpio_init_struct.

Table 390. 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);
```

5.14.4 gpio_input_data_bit_read function

The table below describes the function gpio_input_data_bit_read.

Table 391. 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: selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG or GPIOH.
Input parameter 2	pins: indicates the GPIO pins; refer to *gpio_pins* 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);
```

5.14.5 gpio_input_data_read function

The table below describes the function gpio_input_data_read.

Table 392. 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: selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG or GPIOH.
Output parameter	NA
Return value	Return GPIO input port status
Required preconditions	NA
Called functions	NA

Example:

```
gpio_input_data_read(GPIOA);
```

5.14.6 gpio_output_data_bit_read function

The table below describes the function gpio_output_data_bit_read.

Table 393. 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: selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG or GPIOH.
Input parameter 2	pins: indicates the GPIO pins; refer to *gpio_pins* 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);
```

5.14.7 gpio_output_data_read function

The table below describes the function gpio_output_data_read.

Table 394. 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: selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG or GPIOH.
Output parameter	NA
Return value	Read GPIO output port status
Required preconditions	NA
Called functions	NA

Example:

```
gpio_output_data_read(GPIOA);
```

5.14.8 gpio_bits_set function

The table below describes the function gpio_bits_set.

Table 395. 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: selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG or GPIOH.
Input parameter 2	pins: indicates the GPIO pins; refer to *gpio_pins* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
gpio_bits_set(GPIOA, GPIO_PINS_0);
```

5.14.9 gpio_bits_reset function

The table below describes the function gpio_bits_reset.

Table 396. 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: selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG or GPIOH.
Input parameter 2	pins: indicates the GPIO pins; refer to *gpio_pins* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
gpio_bits_reset(GPIOA, GPIO_PINS_0);
```

5.14.10 gpio_bits_write function

The table below describes the function gpio_bits_write.

Table 397. gpio_bits_write function

Name	Description
Function name	gpio_bits_write
Function prototype	void gpio_bits_write(gpio_type *gpio_x, uint16_t pins, confirm_state bit_state);
Function description	Write GPIO pins
Input parameter 1	gpio_x: selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG or GPIOH.
Input parameter 2	pins: indicates the GPIO pins; refer to *gpio_pins* for details.
Input parameter 3	bit_state: GPIO pin value to be written; it can be 1 (TRUE) or 0 (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
gpio_bits_write(GPIOA, GPIO_PINS_0, TRUE);
```

5.14.11 gpio_bits_toggle function

The table below describes the function gpio_bits_toggle.

Table 398. gpio_bits_toggle function

Name	Description
Function name	gpio_bits_toggle
Function prototype	void gpio_bits_toggle(gpio_type *gpio_x, uint16_t pins);
Function description	Toggle GPIO pins
Input parameter 1	gpio_x: selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG or GPIOH.
Input parameter 2	pins: indicates the GPIO pins; refer to *gpio_pins* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
gpio_bits_toggle (GPIOA, GPIO_PINS_0);
```

5.14.12 gpio_port_write function

The table below describes the function gpio_port_write.

Table 399. 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: selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG or GPIOH.
Input parameter 2	port_value: the port value to be written; it can be 0x0000~0xFFFF.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
gpio_port_write(GPIOA, 0xFFFF);
```

5.14.13 gpio_pin_wp_config function

The table below describes the function gpio_pin_wp_config.

Table 400. 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: selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG or GPIOH.
Input parameter 2	pins: indicates the GPIO pins; refer to *gpio_pins* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
gpio_pin_wp_config(GPIOA, GPIO_PINS_0);
```

5.14.14 gpio_pins_huge_driven_config function

The table below describes the function gpio_pins_huge_driven_config.

Table 401. 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: selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG or GPIOH.
Input parameter 2	pins: indicates the GPIO pins; refer to *gpio_pins* for details.
Input parameter 3	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
gpio_pins_huge_driven_config(GPIOA, GPIO_PINS_0, TRUE);
```

5.14.15 gpio_pin_mux_config function

The table below describes the function gpio_pin_mux_config.

Table 402. 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: selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG or GPIOH
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

GPIO_MUX_8
GPIO_MUX_9
GPIO_MUX_10
GPIO_MUX_11
GPIO_MUX_12
GPIO_MUX_13
GPIO_MUX_14
GPIO_MUX_15

Example:

```
gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE0, GPIO_MUX_0);
```

5.15 I2C interface

The I2C register structure i2c_type is defined in the “at32f435_437_i2c.h”.

```
/**
  * @brief type define i2c register all
  */
typedef struct
{


} i2c_type;
```

The table below gives a list of the I2C registers.

Table 403. Summary of I2C registers

Register	Description
ctrl1	I2C control register 1
ctrl2	I2C control register 2
oaddr1	I2C Own address register 1
oaddr2	I2C Own address register 2
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 I2C library functions.

Table 404. Summary of I2C library functions

Function name	Description
i2c_reset	I2C peripheral reset
i2c_init	Initialize I2C, set bus speed and digital filter
i2c_own_address1_set	Set I2C own address 1
i2c_own_address2_set	Set I2C own address 2
i2c_own_address2_enable	Enable I2C own address 2

i2c_smbus_enable	Enable SMBus mode
i2c_enable	Enable I2C
i2c_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 salve 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_detcet_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 405. I2C application-layer library functions

Function name	Description
i2c_config	I2C application initialization
i2c_lowlevel_init	I2C low-layer initialization
i2c_wait_end	I2C wait data transmit complete
i2c_wait_flag	I2C wait flag
i2c_master_transmit	I2C master transmits data (polling mode)
i2c_master_receive	I2C master receives data (polling mode)
i2c_slave_transmit	I2C slave transmits data (polling mode)

i2c_slave_receive	I2C slave receives data (polling mode)
i2c_master_transmit_int	I2C master transmits data (interrupt mode)
i2c_master_receive_int	I2C master receives data (interrupt mode)
i2c_slave_transmit_int	I2C slave transmits data (interrupt mode)
i2c_slave_receive_int	I2C slave receives data (interrupt mode)
i2c_master_transmit_dma	I2C master transmits data (DMA mode)
i2c_master_receive_dma	I2C master receives data (DMA mode)
i2c_slave_transmit_dma	I2C slave transmits data (DMA mode)
i2c_slave_receive_dma	I2C slave receives data (DMA mode)
i2c_smbus_master_transmit	SMBus master transmits data (polling mode)
i2c_smbus_master_receive	SMBus master receives data (polling mode)
i2c_smbus_slave_transmit	SMBus slave transmits data (polling mode)
i2c_smbus_slave_receive	SMBus slave receives data (polling mode)
i2c_memory_write	I2C writes data to EEPROM (polling mode)
i2c_memory_write_int	I2C writes data to EEPROM (interrupt mode)
i2c_memory_write_dma	I2C writes data to EEPROM (DMA mode)
i2c_memory_read	I2C reads from EEPROM (polling mode)
i2c_memory_read_int	I2C reads from EEPROM (interrupt mode)
i2c_memory_read_dma	I2C reads from EEPROM (DMA mode)
i2c_evt_irq_handler	I2C event interrupt function
i2c_err_irq_handler	I2C error interrupt function
i2c_dma_tx_irq_handler	I2C DMA Tx interrupt function
i2c_dma_rx_irq_handler	I2C DMA Rx interrupt function

5.15.1 i2c_reset function

The table below describes the function i2c_reset.

Table 406. i2c_reset function

Name	Description
Function name	i2c_reset
Function prototype	void i2c_reset(i2c_type *i2c_x);
Function description	Reset all I2C registers to their initial values through CRM (Clock and reset management)
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	void crm_periph_reset(crm_periph_reset_type value, confirm_state new_state)

Example:

```
i2c_reset(I2C1);
```

5.15.2 i2c_init function

The table below describes the function i2c_init.

Table 407. 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 I2C bus speed and digital filter
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	dfilters: digital filter, ranging from 0x00 to 0x0F 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 I2C communication speed. This value can be calculated through “Artery_I2C_Timing_Configuration” defined in *AN0091_AT32F435_437_I2C_Application_Note*.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_init(I2C1, 0x0F, 0x80504C4E);
```

5.15.3 i2c_own_address1_set function

The table below describes the function i2c_own_address1_set.

Table 408. 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 address1
Input parameter 1	mode: Own address 1 address mode Refer to the following “mode” descriptions 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);
```

5.15.4 i2c_own_address2_set function

The table below describes the function i2c_own_address2_set.

Table 409. 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 Note: only 7-bit address is supported, not 10-bit address mode.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	address: own address 2
Input parameter 3	mask: own address 2 bit mask 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]
I2C_ADDR2_MASK07:	All non-I2C reserved addresses would respond

Example:

```
i2c_own_address2_set(I2C1, 0xB0, I2C_ADDR2_NOMASK);
```

5.15.5 i2c_own_address2_enable function

The table below describes the function i2c_own_address2_enable.

Table 410. i2c_own_address2_enable function

Name	Description
Function name	i2c_own_address2_enable
Function prototype	void i2c_own_address2_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable own address 2. The address becomes active only after it is enabled. Note that this function should be used in conjunction with the i2c_own_address2_set.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	new_state: address 2 status 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);
```

5.15.6 i2c_smbus_enable function

The table below describes the function i2c_smbus_enable.

Table 411. 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 I2C mode.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	mode: SMBus mode selection Refer to the following “mode” descriptions for details.
Input parameter 3	new_state: SMBus mode status 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);
```

5.15.7 i2c_enable function

The table below describes the function i2c_enable。

Table 412. i2c_enable function

Name	Description
Function name	i2c_enable
Function prototype	void i2c_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable I2C peripheral
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	new_state: indicates I2C status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_enable(I2C1, TRUE);
```

5.15.8 i2c_clock_stretch_enable function

The table below describes the function i2c_clock_stretch_enable.

Table 413. i2c_clock_stretch_enable function

Name	Description
Function name	i2c_clock_stretch_enable
Function prototype	void i2c_clock_stretch_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable clock stretching capability. This function is applicable to slave mode only. In most cases, enabling the clock stretching mode is recommended in order to prevent slave from having no sufficient time to receive or send data due to slow process speed, which causes a loss of data. It should be noted that the host must be able to support clock stretching function before using this mode by slave. For example, some hosts based on IO analog are not equipped with the clock stretching capability.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	new_state: indicates clock stretching status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_clock_stretch_enable(I2C1, TRUE);
```

5.15.9 i2c_ack_enable function

The table below describes the function i2c_ack_enable.

Table 414. 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	This function is used to enable ACK or NACK of each byte in master and slave mode. For ACK information on I2C communication protocol, refer to I2C protocol or AT32 reference manual.
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	new_state: indicates ACK response status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_ack_enable(I2C1, TRUE);
```

5.15.10 i2c_addr10_mode_enable function

The table below describes the function i2c_addr10_mode_enable.

Table 415. 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	i2c_x: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	new_state: 10-bit address mode enable state This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_addr10_mode_enable(I2C1, TRUE);
```

5.15.11 i2c_transfer_addr_set function

The table below describes the function i2c_transfer_addr_set.

Table 416. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
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);
```

5.15.12 i2c_transfer_addr_get function

The table below describes the function i2c_transfer_addr_get.

Table 417. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Output parameter	NA
Return value	uint16_t: slave address sent from master
Required preconditions	NA
Called functions	NA

Example:

```
i2c_transfer_addr_get(I2C1);
```

5.15.13 i2c_transfer_dir_set function

The table below describes the function i2c_transfer_dir_set.

Table 418. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	direction: data transfer direction 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);
```

5.15.14 i2c_transfer_dir_get function

The table below describes the function i2c_transfer_dir_get.

Table 419. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Output parameter	NA
Return value	i2c_transfer_dir_type: slave data transfer direction 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);
```

5.15.15 i2c_matched_addr_get function

The table below describes the function i2c_matched_addr_get.

Table 420. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Output parameter	NA
Return value	uint8_t: slave matched address
Required preconditions	NA
Called functions	NA

Example:

```
i2c_matched_addr_get(I2C1);
```

5.15.16 i2c_auto_stop_enable function

The table below describes the function i2c_auto_stop_enable.

Table 421. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	new_state: auto transmit stop condition enable state 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);
```

5.15.17 i2c_reload_enable function

The table below describes the function i2c_reload_enable.

Table 422. 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

Name	Description
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	new_state: reload mode enable state 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);
```

5.15.18 i2c_cnt_set function

The table below describes the function i2c_cnt_set.

Table 423. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
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);
```

5.15.19 i2c_addr10_header_enable function

The table below describes the function i2c_addr10_header_enable.

Table 424. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	new_state: enable state of auto transmit stop conditions 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);
```

5.15.20 i2c_general_call_enable function

The table below describes the function i2c_general_call_enable.

Table 425. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	new_state: Broadcast address enable state 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);
```

5.15.21 i2c_smbus_alert_set function

The table below describes the function i2c_smbus_alert_set.

Table 426. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	level: SMBus alert pin level Refer to the following “level” descriptions for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

level

SMBus alert pin level

I2C_SMBUS_ALERT_LOW: SMBus alert pin output low

I2C_SMBUS_ALERT_HIGH: SMBus alert pin output high

Example:

```
i2c_smbus_alert_set(I2C1, I2C_SMBUS_ALERT_LOW);
```

5.15.22 i2c_slave_data_ctrl_enable function

The table below describes the function i2c_slave_data_ctrl_enable.

Table 427. i2c_slave_data_ctrl_enable 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	new_state: enable state 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);
```

5.15.23 i2c_pec_calculate_enable function

The table below describes the function i2c_pec_calculate_enable.

Table 428. i2c_pec_calculate_enable function

Name	Description
Function name	i2c_pec_calculate_enable
Function prototype	void i2c_pec_calculate_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable PEC calculation
Input parameter 1	i2c_x: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	new_state: PEC calculation state 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);
```

5.15.24 i2c_pec_transmit_enable function

The table below describes the function i2c_pec_transmit_enable.

Table 429. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	new_state: PEC transmit enable state 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);
```

5.15.25 i2c_pec_value_get function

The table below describes the function i2c_pec_value_get.

Table 430. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Output parameter	uint8_t: current PEC value
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
pec_value = i2c_pec_value_get(I2C1);
```

5.15.26 i2c_timeout_set function

The table below describes the function i2c_timeout_set.

Table 431. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
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, 0x0FFF);
```

5.15.27 i2c_timeout_detcet_set function

The table below describes the function i2c_timeout_detcet_set.

Table 432. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	mode: level detect mode 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);
```

5.15.28 i2c_timeout_enable function

The table below describes the function i2c_timeout_enable.

Table 433. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	new_state: level timeout detect enable state 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);
```

5.15.29 i2c_ext_timeout_set function

The table below describes the function i2c_ext_timeout_set.

Table 434. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
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);
```

5.15.30 i2c_ext_timeout_enable function

The table below describes the function i2c_ext_timeout_enable.

Table 435. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	new_state: cumulative clock stretching timeout enable state 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);
```

5.15.31 i2c_interrupt_enable function

The table below describes the function i2c_interrupt_enable.

Table 436. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	source: interrupt sources Refer to the following “source” descriptions for details.
Input parameter 3	new_state: interrupt enable state This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

source

Select an 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);
```

5.15.32 i2c_interrupt_get function

The table below describes the function i2c_interrupt_get.

Table 437. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	source: interrupt source Refer to the following “source” descriptions for details.
Output parameter	flag_status: flag status This parameter can be SET or RESET.
Return value	NA
Required preconditions	NA
Called functions	NA

source

Select an 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_get(I2C1, I2C_TD_INT, TRUE);
```

5.15.33 i2c_dma_enable function

The table below describes the function i2c_dma_enable.

Table 438. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	dma_req: DMA request Refer to the following “dma_req” descriptions for details.
Input parameter 3	new_state: DMA enable state This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
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);
```

5.15.34 i2c_transmit_set function

The table below describes the function i2c_transmit_set.

Table 439. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
	address: slave address
	cnt: count of data to send/receive
Input parameter 2	rld_stop: reload mode and STOP condition generation mode Refer to the following “rld_stop” descriptions for details.
Input parameter 3	start: set START condition generation mode 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 >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);
```

5.15.35 i2c_start_generate function

The table below describes the function i2c_start_generate.

Table 440. i2c_start_generate 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_start_generate(I2C1);
```

5.15.36 i2c_stop_generate function

The table below describes the function i2c_stop_generate.

Table 441. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_stop_generate(I2C1);
```

5.15.37 i2c_data_send function

The table below describes the function i2c_data_send.

Table 442. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
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);
```

5.15.38 i2c_data_receive function

The table below describes the function i2c_data_receive.

Table 443. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Output parameter	uint8_t: data to be received
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
data_value = i2c_data_receive(I2C1);
```

5.15.39 i2c_flag_get function

The table below describes the function i2c_flag_get.

Table 444. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.

Name	Description
Input parameter 2	flag: the selected flag Refer to the following “flag” descriptions for details.
Output parameter	NA
Return value	flag_status: flag status This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

flag

This 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);
```

5.15.40 i2c_interrupt_flag_get function

The table below describes the function i2c_interrupt_flag_get.

Table 445. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	flag: the selected flag 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);
```

5.15.41 i2c_flag_clear function

The table below describes the function i2c_flag_clear。

Table 446. 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: selected I2C peripheral This parameter can be I2C1, I2C2 or I2C3.
Input parameter 2	flag: the selected flag 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);
```

5.15.42 i2c_config function

The table below describes the function i2c_config.

Table 447. i2c_config function

Name	Description
Function name	i2c_config
Function prototype	void i2c_config(i2c_handle_type* hi2c);
Function description	I2C initialization function used to initialize I2C. Call the function i2c_lowlevel_init() to initialize I2C peripherals, GPIO, DMA, interrupts and others.
Input parameter 1	hi2c: i2c_handle_type pointer Refet to *i2c_handle_type*.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	void i2c_lowlevel_init(i2c_handle_type* hi2c);

i2c_handle_type* hi2c

The i2c_handle_type is defined in the i2c_application.h.

```
typedef struct
{
    i2c_type                *i2cx;
    uint8_t                 *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 I2C peripheral from I2C1, I2C2 or I2C3.

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

I2C 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 occurs

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);
```

5.15.43 i2c_lowlevel_init function

The table below describes the function i2c_lowlevel_init.

Table 448. i2c_lowlevel_init function

Name	Description
Function name	i2c_lowlevel_init
Function prototype	void i2c_lowlevel_init(i2c_handle_type* hi2c);
Function description	I2C lower-level initialization callback function. It is called in the i2c_config to initialize I2C peripherals, GPIO, DMA, interrupts, etc. It requires users to implement I2C initialization inside the function.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type*.
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 I2C2 initialization
  }
}
```

5.15.44 i2c_wait_end function

The table below describes the function i2c_wait_end.

Table 449. 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 Refer to *i2c_handle_type*.
Input parameter 2	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to Section 5.15.42 for details.

Name	Description
Required preconditions	NA
Called functions	NA

Example:

```
if (i2c_master_transmit_dma(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF) != I2C_OK)
{
   error_handler(i2c_status);
}

/* wait for the end of transfer */
if(i2c_wait_end(&hi2c, 0xFFFFFFFF) != I2C_OK)
{
   error_handler(i2c_status);
}
```

5.15.45 i2c_wait_flag function

The table below describes the function i2c_wait_flag.

Table 450. 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 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 Refer to *i2c_handle_type*.
Input parameter 2	flag: the selected flag 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 Refer to the following “event_check” descriptions for details.
Input parameter 4	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to Section 5.15.42 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);
```

5.15.46 i2c_master_transmit function

The table below describes the function i2c_master_transmit.

Table 451. 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 I2C transfer ends after the function is executed.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type*.
Input parameter 2	address: slave address
Input parameter 3	pdata: array address of to-be-sent data
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 Refer to Section 5.15.42 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_transmit(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

5.15.47 i2c_master_receive function

The table below describes the function i2c_master_receive.

Table 452. i2c_master_receive function

Name	Description
Function name	i2c_master_receive
Function prototype	i2c_status_type i2c_master_receive(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Master receives data (polling mode). This function is a blocking type. After the execution is done, so does I2C transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type*.
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 Refer to Section 5.15.42 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_receive(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

5.15.48 i2c_slave_transmit function

The table below describes the function i2c_slave_transmit.

Table 453. 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 does the I2C transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type*.
Input parameter 2	pdata: array address of to-be-sent data
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 Refer to Section 5.15.42 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_transmit(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

5.15.49 i2c_slave_receive function

The table below describes the function i2c_slave_receive.

Table 454. 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 does the I2C transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type*.
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 Refer to Section 5.15.42 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_receive(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

5.15.50 i2c_master_transmit_int function

The table below describes the function i2c_master_transmit_int.

Table 455. 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, I2C 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 Refer to *i2c_handle_type*.
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

Name	Description
	Refer to Section 5.15.42 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_transmit_int(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

5.15.51 i2c_master_receive_int function

The table below describes the function i2c_master_receive_int.

Table 456. 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 I2C transfer has not completed yet. So in this case, it is possible to call i2c_wait_end() to wait for the end of transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type*.
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 Refer to Section 5.15.42 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_receive_int(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

5.15.52 i2c_slave_transmit_int function

The table below describes the function i2c_slave_transmit_int.

Table 457. i2c_slave_transmit_int function

Name	Description
Function name	i2c_slave_transmit_int
Function prototype	i2c_status_type i2c_slave_transmit_int(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave sends data (through interrupt mode). This function operates in non-blocking mode. In other words, after the function is executed, the I2C 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.

Name	Description
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type*.
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 Refer to Section 5.15.42 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_transmit_int(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

5.15.53 i2c_slave_receive_int function

The table below describes the function i2c_slave_receive_int.

Table 458. i2c_slave_receive_int function

Name	Description
Function name	i2c_slave_receive_int
Function prototype	i2c_status_type i2c_slave_receive_int(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave receives data (through interrupt mode). This function is a non-blocking type. In other words, after the function is executed, the I2C 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 Refer to *i2c_handle_type*.
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 Refer to Section 5.15.42 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_receive_int(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

5.15.54 i2c_master_transmit_dma function

The table below describes the function i2c_master_transmit_dma.

Table 459. 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	Slave sends data (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I2C 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 Refer to *i2c_handle_type*.
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 Refer to Section 5.15.42 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_transmit_dma(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

5.15.55 i2c_master_receive_dma function

The table below describes the function i2c_master_receive_dma.

Table 460. 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 I2C 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 Refer to *i2c_handle_type*.
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

Name	Description
Return value	i2c_status_type: error code Refer to Section 5.15.42 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_receive_dma(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

5.15.56 i2c_slave_transmit_dma function

The table below describes the function i2c_slave_transmit_dma.

Table 461. 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 I2C 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 Refer to *i2c_handle_type*.
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 Refer to Section 5.15.42 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_transmit_dma(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

5.15.57 i2c_slave_receive_dma function

The table below describes the function i2c_slave_receive_dma.

Table 462. i2c_slave_receive_dma function

Name	Description
Function name	i2c_slave_receive_dma
Function prototype	i2c_status_type i2c_slave_receive_dma(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave receives data (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I2C 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.

Name	Description
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type*.
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 Refer to Section 5.15.42 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_receive_dma(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

5.15.58 i2c_smbus_master_transmit function

The table below describes the function i2c_smbus_master_transmit.

Table 463. 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 does the data transfer. It is mainly used for PEC transmission and reception.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type*.
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 Refer to Section 5.15.42 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_smbus_master_transmit(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

5.15.59 i2c_smbus_master_receive function

The table below describes the function i2c_smbus_master_receive.

Table 464. 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 does the data transfer. It is mainly used for PEC transmission and reception.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type*.
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 Refer to Section 5.15.42 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_smbus_master_receive(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

5.15.60 i2c_smbus_slave_transmit function

The table below describes the function i2c_smbus_slave_transmit.

Table 465. 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 does the data transfer. It is mainly used for PEC transmission and reception.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type*.
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 Refer to Section 5.15.42 for details.
Required preconditions	NA

Name	Description
Called functions	NA

Example:

```
i2c_smbus_slave_transmit(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

5.15.61 i2c_smbus_slave_receive function

The table below describes the function i2c_smbus_slave_receive.

Table 466. 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 Refer to *i2c_handle_type*.
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 Refer to Section 5.15.42 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_smbus_slave_receive(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

5.15.62 i2c_memory_write function

The table below describes the function i2c_memory_write.

Table 467. 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 does the I2C transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type*.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the following “mem_address_width” descriptions for details.
Input parameter 3	address: EEPROM address

Name	Description
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 Refer to Section 5.15.42 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);
```

5.15.63 i2c_memory_write_int function

The table below describes the function i2c_memory_write_int.

Table 468. i2c_memory_write_int function

Name	Description
Function name	i2c_memory_write_int
Function prototype	i2c_status_type i2c_memory_write_int(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Write EEPROM (through interrupt mode). This function is a non-blocking type. In other words, after the function is executed, the I2C 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 Refer to *i2c_handle_type*.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the following “mem_address_width” descriptions 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 Refer to Section 5.15.42 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);
```

5.15.64 i2c_memory_write_dma function

The table below describes the function i2c_memory_write_dma.

Table 469. i2c_memory_write_dma function

Name	Description
Function name	i2c_memory_write_dma
Function prototype	i2c_status_type i2c_memory_write_dma(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Write EEPROM (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I2C 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 Refer to *i2c_handle_type*.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the following “mem_address_width” descriptions 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 Refer to Section 5.15.42 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);
```

5.15.65 i2c_memory_read function

The table below describes the function i2c_memory_read.

Table 470. i2c_memory_read function

Name	Description
Function name	i2c_memory_read
Function prototype	i2c_status_type i2c_memory_read(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Read EEPROM (through polling mode). This function is a blocking type. In other words, after the function execution is done, so does the data transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type*.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the following “mem_address_width” descriptions 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 Refer to Section 5.15.42 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);
```

5.15.66 i2c_memory_read_int function

The table below describes the function i2c_memory_read_int.

Table 471. i2c_memory_read_int function

Name	Description
Function name	i2c_memory_read_int
Function prototype	i2c_status_type i2c_memory_read_int(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Read EEPROM (through interrupt mode). This function is a non-blocking type. In other words, after the function is executed, the I2C 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 Refer to *i2c_handle_type*.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the following “mem_address_width” descriptions 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 Refer to Section 5.15.42 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);
```

5.15.67 i2c_memory_read_dma function

The table below describes the function i2c_memory_read_dma.

Table 472. i2c_memory_read_dma function

Name	Description
Function name	i2c_memory_read_dma
Function prototype	i2c_status_type i2c_memory_read_dma(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Read EEPROM (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I2C transfer has not completed yet.

Name	Description
	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 Refer to *i2c_handle_type*.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the following “mem_address_width” descriptions 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 Refer to Section 5.15.42 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);
```

5.15.68 i2c_evt_irq_handler function

The table below describes the function i2c_evt_irq_handler.

Table 473. 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 I2C event interrupt.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type*.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
void I2C1_EVT_IRQHandler(void)
{
  i2c_evt_irq_handler(&hi2c);
}
```

5.15.69 i2c_err_irq_handler function

The table below describes the function i2c_err_irq_handler.

Table 474. 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 I2C error interrupt.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to *i2c_handle_type*.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
void I2C1_ERR_IRQHandler(void)
{
  i2c_err_irq_handler(&hi2c);
}
```

5.15.70 i2c_dma_tx_irq_handler function

The table below describes the function i2c_dma_tx_irq_handler.

Table 475. 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 Refer to *i2c_handle_type*.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
void DMA1_Channel6_IRQHandler(void)
{
  i2c_dma_rx_irq_handler(&hi2c);
}
```

5.15.71 i2c_dma_rx_irq_handler function

The table below describes the function i2c_dma_rx_irq_handler.

Table 476. 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 Refer to *i2c_handle_type*.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
void DMA1_Channel7_IRQHandler(void)
{
  i2c_dma_tx_irq_handler(&hi2c);
}
```

5.16 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 477. 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 478. Summary of NVIC library functions

Function name	Description
nvic_system_reset	System software reset
nvic_irq_enable	NVIC interrupt enable and priority configuration
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

5.16.1 nvic_system_reset function

The table below describes the function nvic_system_reset.

Table 479. 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

Name	Description
Required preconditions	NA
Called functions	NVIC_SystemReset()

Example:

```
/* system reset */
nvic_system_reset();
```

5.16.2 nvic_irq_enable function

The table below describes the function nvic_irq_enable.

Table 480. 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 Refer to *irqn*.
Input parameter 2	preempt_priority: set preemption priority This parameter cannot be greater than the highest preemption priority defined in the NVIC_PRIORITY_GROUP_x.
Input parameter 3	sub_priority: set response priority This parameter cannot be greater than the highest response priority defined in the NVIC_PRIORITY_GROUP_x.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NVIC_SetPriority() NVIC_EnableIRQ()

irqn

The irqn is used to select interrupt vectors, including

WWDT_IRQn: Window timer interrupt

PVM_IRQn: PVM interrupt linked to EXINT

……

DMA2_Channel6_IRQn: DMA2 channel 6 global interrupt

DMA2_Channel7_IRQn: DMA2 channel 7 global interrupt

Example:

```
/* enable nvic irq */
nvic_irq_enable(ADC1_2_3_IRQn, 0, 0);
```

5.16.3 nvic_irq_disable function

The table below describes the function nvic_irq_disable.

Table 481. nvic_irq_disable function

Name	Description
Function name	nvic_irq_disable
Function prototype	void nvic_irq_disable(IRQn_Type irqn)
Function description	NVIC interrupt enable
Input parameter	irqn: select interrupt vector. Refer to *irqn*.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NVIC_DisableIRQ()

Example:

```
/* disable nvic irq */
nvic_irq_disable(ADC1_2_3_IRQn);
```

5.16.4 nvic_priority_group_config function

The table below describes the function nvic_priority_group_config.

Table 482. nvic_priority_group_config function

Name	Description
Function name	nvic_priority_group_config
Function prototype	void nvic_priority_group_config(nvic_priority_group_type priority_group)
Function description	NVIC interrupt priority grouping configuration
Input parameter	priority_group: select interrupt priority group This parameter can be any enumerated value in the nvic_priority_group_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NVIC_SetPriorityGrouping()

priority_group

The 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 3 (4 bits for preemption priority, and 0 bit for response priority)

Example:

```
/* config nvic priority group */
nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
```

5.16.5 nvic_vector_table_set function

The table below describes the function nvic_vector_table_set.

Table 483. nvic_vector_table_set function

Name	Description
Function name	nvic_vector_table_set
Function prototype	void nvic_vector_table_set(uint32_t base, uint32_t offset)
Function description	Set NVIC interrupt vector table base address and offset address
Input parameter 1	base: base address of interrupt vector table The base address can be set in RAM or FLASH.
Input parameter 2	offset: offset address of interrupt vector table This parameter defines the start address of interrupt vector table, so it must be set to a multiple of 0x200.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

base

The 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);
```

5.16.6 nvic_lowpower_mode_config function

The table below describes the function nvic_lowpower_mode_config.

Table 484. nvic_lowpower_mode_config function

Name	Description
Function name	nvic_lowpower_mode_config
Function prototype	void nvic_lowpower_mode_config(nvic_lowpower_mode_type lp_mode, confirm_state new_state)
Function description	Configure NVIC low-power mode
Input parameter 1	lp_mode: select low-power modes This parameter can be any enumerated value in the nvic_lowpower_mode_type.
Input parameter 2	new_state: indicates the pre-configured status of battery powered domain This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

lp_mode

The 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);
```

5.17 Power controller (PWC)

The PWC register structure pwc_type is defined in the “at32f435_437_pwc.h”.

```
/**
  * @brief type define pwc register all
  */
typedef struct
{
      ……
} pwc_type;
```

The table below gives a list of the PWC registers.

Table 485. Summary of PWC registers

Register	Description
ctrl	Power control register
ctrlsts	Power control/status register
ldoov	LDO calibration register

The table below gives a list of PWC library functions.

Table 486. Summary of PWC library functions

Function name	Description
pwc_reset	Reset PWC registers to their reset values
pwc_battery_powered_domain_access	Enable battery powered domain access
pwc_pvm_level_select	Select PVM threshold
pwc_power_voltage_monitor_enable	Enable voltage monitor
pwc_wakeup_pin_enable	Enable standby-mode wakeup pin
pwc_flag_clear	Clear flag
pwc_flag_get	Get flag status
pwc_sleep_mode_enter	Enter Sleep mode
pwc_deep_sleep_mode_enter	Enter Deepsleep mode
pwc_voltage_regulate_set	Select voltage regulator status in Deepsleep mode
pwc_standby_mode_enter	Enter Standby mode
pwc_ldo_output_voltage_set	Set LDO output voltage

5.17.1 pwc_reset function

The table below describes the function pwc_reset.

Table 487. 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();
```

5.17.2 pwc_battery_powered_domain_access function

The table below describes the function pwc_battery_powered_domain_access.

Table 488. 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 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.

5.17.3 pwc_pvm_level_select function

The table below describes the function pwc_pvm_level_select.

Table 489. pwc_pvm_level_select function

Name	Description
Function name	pwc_pvm_level_select
Function prototype	void pwc_pvm_level_select(pwc_pvm_voltage_type pvm_voltage)
Function description	Select PVM threshold
Input parameter	pvm_voltage: indicates the selected PVM threshold This parameter can be any enumerated value in the pwc_pvm_voltage_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pvm_voltage

The pvm_voltage is used to select a PVM threshold from the optional parameters below:

PWC_PVM_VOLTAGE_2V3: PVM threshold is 2.3 V
PWC_PVM_VOLTAGE_2V4: PVM threshold is 2.4 V
PWC_PVM_VOLTAGE_2V5: PVM threshold is 2.5 V
PWC_PVM_VOLTAGE_2V6: PVM threshold is 2.6 V
PWC_PVM_VOLTAGE_2V7: PVM threshold is 2.7 V
PWC_PVM_VOLTAGE_2V8: PVM threshold is 2.8 V
PWC_PVM_VOLTAGE_2V9: PVM threshold is 2.9 V

Example:

```
/* set the threshold voltage to 2.9v */
pwc_pvm_level_select(PWC_PVM_VOLTAGE_2V9);
```

5.17.4 pwc_power_voltage_monitor_enable function

The table below describes the function pwc_power_voltage_monitor_enable.

Table 490. pwc_power_voltage_monitor_enable function

Name	Description
Function name	pwc_power_voltage_monitor_enable
Function prototype	void pwc_power_voltage_monitor_enable(confirm_state new_state)
Function description	Enable power voltage monitor (PVM)
Input parameter	new_state: indicates the pre-configured status of PVM This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable power voltage monitor */
pwc_power_voltage_monitor_enable(TRUE);
```

5.17.5 pwc_wakeup_pin_enable function

The table below describes the function pwc_wakeup_pin_enable.

Table 491. 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 This parameter can be any pin that is capable of waking up from Standby mode.
Input parameter 2	new_state: indicates the pre-configured status of Standby wakeup pins This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pin_num

The 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 2 (corresponding GPIO is PC13)

Example:

```
/* enable wakeup pin - pa0 */
pwc_wakeup_pin_enable(PWC_WAKEUP_PIN_1, TRUE);
```

5.17.6 pwc_flag_clear function

The table below describes the function pwc_flag_clear.

Table 492. 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 Refer to *pwc_flag*.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pwc_flag

The 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 flag

PWC_PVM_OUTPUT_FLAG: PVM output flag (this parameter cannot be cleared by software)

Example:

```
/* wakeup event flag clear */
pwc_flag_clear(PWC_WAKEUP_FLAG);
```

5.17.7 pwc_flag_get function

The table below describes the function pwc_flag_get.

Table 493. 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 *pwc_flag*.
Output parameter	NA
Return value	flag_status: indicates flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

```
/* check if wakeup event flag is set */
if(pwc_flag_get(PWC_WAKEUP_FLAG) != RESET)
```

5.17.8 pwc_sleep_mode_enter function

The table below describes the function pwc_sleep_mode_enter.

Table 494. pwc_sleep_mode_enter function

Name	Description
Function name	pwc_sleep_mode_enter
Function prototype	void pwc_sleep_mode_enter(pwc_sleep_enter_type pwc_sleep_enter)
Function description	Enter Sleep mode
Input parameter	pwc_sleep_enter: select a command to enter Sleep mode This parameter can be any enumerated value in the pwc_sleep_enter_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pwc_sleep_enter

The 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);
```

5.17.9 pwc_deep_sleep_mode_enter function

The table below describes the function pwc_deep_sleep_mode_enter.

Table 495. pwc_deep_sleep_mode_enter function

Name	Description
Function name	pwc_deep_sleep_mode_enter
Function prototype	void pwc_deep_sleep_mode_enter(pwc_deep_sleep_enter_type pwc_deep_sleep_enter)
Function description	Enter Deepsleep mode
Input parameter	pwc_deep_sleep_enter: select a command to enter Deepsleep mode This parameter can be any enumerated value in the pwc_deep_sleep_enter_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pwc_deep_sleep_enter

The pwc_deep_sleep_enter is used to select a command to enter Deepsleep mode, including:

PWC_DEEP_SLEEP_ENTER_WFI: Enter Deepsleep mode WFI

PWC_DEEP_SLEEP_ENTER_WFE: Enter Deepsleep mode WFE

Example:

```
/* enter deep sleep mode */
pwc_deep_sleep_mode_enter(PWC_DEEP_SLEEP_ENTER_WFI);
```

5.17.10 pwc_voltage_regulate_set function

The table below describes the function pwc_voltage_regulate_set.

Table 496. 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 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

The 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:

```
/* congfig the voltage regulator mode */
pwc_voltage_regulate_set(PWC_REGULATOR_LOW_POWER);
```

5.17.11 pwc_standby_mode_enter function

The table below describes the function pwc_standby_mode_enter.

Table 497. 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();
```

5.17.12 pwc_ldo_output_voltage_set function

The table below describes the function pwc_ldo_output_voltage_set.

Table 498. pwc_ldo_output_voltage_set function

Name	Description
Function name	pwc_ldo_output_voltage_set
Function prototype	pwc_ldo_output_voltage_set(val)
Function description	Set LDO output voltage
Input parameter	Val: LDO output voltage value This parameter can be any value enumerated in the pwc_ldo_output_voltage_type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Val:

Val is used to select LDO output voltage value from the following parameters:

PWC_LDO_OUTPUT_1V1: LDO output voltage 1.1V

PWC_LDO_OUTPUT_1V2: LDO output voltage 1.2V

PWC_LDO_OUTPUT_1V3: LDO output voltage 1.3V

Example:

```
/* reduce ldo before enter deepsleep mode */
pwc_ldo_output_voltage_set(PWC_LDO_OUTPUT_1V1);
```

5.18 Qud-SPI interface (QSPI)

The QSPI register structure qspi_type is defined in the “at32f435_437_qspi.h”.

```
/**
  * @brief type define qspi register all
  */
typedef struct
{
      …
} qspi_type;
```

The table below gives a list of the QSPI registers.

Table 499. Summary of QSPI registers

Register	Description
cmd_w0	Command word 0
cmd_w1	Command word 1
cmd_w2	Command word 2
cmd_w3	Command word 3
ctrl	Control register
actr	AC timing register
fifosts	FIFO status register
ctrl2	Control register 2
cmdsts	Command status register
rsts	Read status register
fsize	Flash size register
xip cmd_w0	XIP command word 0
xip cmd_w1	XIP command word 1
xip cmd_w2	XIP command word 2
xip cmd_w3	XIP command word 3
rev	Revision register
dt	Data port register

The table below gives a list of QSPI library functions.

Table 500. Summary of QSPI library functions

Function name	Description
qspi_encryption_enable	QSPI encryption enable
qspi_sck_mode_set	Configure QSPI clock mode
qspi_clk_division_set	Configure QSPI clock division
qspi_xip_cache_bypass_set	Enable QSPI XIP port cache bypass
qspi_interrupt_enable	Enable QSPI interrupt
qspi_flag_get	Get QSPI flag
qspi_flag_clear	Clear QSPI flag
qspi_dma_rx_threshold_set	Configure QSPI DMA receive FIFO threshold
qspi_dma_tx_threshold_set	Configure QSPI DMA transmit FIFO threshold

Function name	Description
qspi_dma_enable	Enable QSPI DMA
qspi_busy_config	Configure the offset of QSPI busy bit in the status register
qspi_xip_enable	QSPI XIP port enable
qspi_cmd_operation_kick	Start QSPI command operation
qspi_xip_init	QSPI XIP port initialization
qspi_byte_read	QSPI byte read
qspi_half_word_read	QSPI half-word read
qspi_word_read	QSPI word read
qspi_word_write	QSPI word write
qspi_half_word_write	QSPI half-word write
qspi_byte_write	QSPI byte write

5.18.1 qspi_encryption_enable function

The table below describes the function qspi_encryption_enable

Table 501. qspi_encryption_enable function function

Name	Description
Function name	qspi_encryption_enable
Function prototype	void qspi_encryption_enable(qspi_type* qspi_x, confirm_state new_state);
Function description	Enable QSPI encryption. This function is called only when QSPI is in the command port.
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	new_state: encryption status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
qspi_encryption_enable(QSPI1, TRUE);
```

5.18.2 qspi_sck_mode_set function

The table below describes the function qspi_sck_mode_set.

Table 502. qspi_sck_mode_set function

Name	Description
Function name	qspi_sck_mode_set
Function prototype	void qspi_sck_mode_set(qspi_type* qspi_x, qspi_clk_mode_type new_mode);
Function description	Configure QSPI clock mode
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	new_mode: clock mode
Output parameter	NA

Name	Description
Return value	NA
Required preconditions	NA
Called functions	NA

qspi_clk_mode_type

QSPI_SCK_MODE_0: QSPI clock mode 0

QSPI_SCK_MODE_3: QSPI clock mode 3

Example:

```
qspi_sck_mode_set(QSPI1, QSPI_SCK_MODE_0);
```

5.18.3 qspi_clk_division_set function

The table below describes the function qspi_clk_division_set.

Table 503. qspi_clk_division_set function

Name	Description
Function name	qspi_clk_division_set
Function prototype	void qspi_clk_division_set (qspi_type* qspi_x, qspi_clk_div_type new_clkdiv);
Function description	Configure QSPI clock division
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	new_clkdiv: clock division
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

qspi_clk_div_type

QSPI_CLK_DIV_2: Divided by 2

QSPI_CLK_DIV_4: Divided by 4

QSPI_CLK_DIV_6: Divided by 6

QSPI_CLK_DIV_8: Divided by 8

QSPI_CLK_DIV_3: Divided by 3

QSPI_CLK_DIV_5: Divided by 5

QSPI_CLK_DIV_10: Divided by 10

QSPI_CLK_DIV_12: Divided by 12

Example:

```
qspi_clk_division_set(QSPI1, QSPI_CLK_DIV_2);
```

5.18.4 qspi_xip_cache_bypass_set function

The table below describes the function qspi_xip_cache_bypass_set.

Table 504. qspi_xip_cache_bypass_set function

Name	Description
Function name	qspi_xip_cache_bypass_set
Function prototype	void qspi_xip_cache_bypass_set(qspi_type* qspi_x, confirm_state new_state);
Function description	Configure QSPI XIP port cache bypass
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	new_state: bypass status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
qspi_xip_cache_bypass_set(QSPI1, TRUE);
```

5.18.5 qspi_interrupt_enable function

The table below describes the function qspi_interrupt_enable.

Table 505. qspi_interrupt_enable function

Name	Description
Function name	qspi_interrupt_enable
Function prototype	void qspi_interrupt_enable(qspi_type* qspi_x, confirm_state new_state);
Function description	Configure QSPI interrupts
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	new_state: interrupt status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
qspi_interrupt_enable(QSPI1, TRUE);
```

5.18.6 qspi_interrupt_flag_get function

The table below describes the function qspi_flag_get.

Table 506. qspi_flag_get function

Name	Description
Function name	qspi_interrupt_flag_get
Function prototype	flag_status qspi_interrupt_flag_get(qspi_type* qspi_x, uint32_t flag);
Function description	Get QSPI interrupt flag status
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	flag: only QSPI_CMDSTS_FLAG is valid
Output parameter	NA
Return value	flag_status: SET or RESET.
Required preconditions	NA
Called functions	NA

qspi_flag

QSPI_CMDSTS_FLAG: QSPI command complete flag

Example:

```
flag_status status;
status = qspi_flag_get(QSPI_CMDSTS_FLAG);
```

5.18.7 qspi_flag_get function

The table below describes the function qspi_flag_get.

Table 507. qspi_flag_get function

Name	Description
Function name	qspi_flag_get
Function prototype	flag_status qspi_flag_get(qspi_type* qspi_x, uint32_t flag);
Function description	Get flag status
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	flag: flag selection
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

qspi_flag

Get a QSPI status flag

QSPI_CMDSTS_FLAG: QSPI command complete flag

QSPI_RXFIFORDY_FLAG: QSPI receive FIFO ready flag

QSPI_TXFIFORDY_FLAG: QSPI transmit FIFO ready flag

Example:

```
flag_status status;
status = qspi_flag_get(QSPI_CMDSTS_FLAG);
```

5.18.8 qspi_flag_clear function

The table below describes the function qspi_flag_clear.

Table 508. qspi_flag_clear function

Name	Description
Function name	qspi_flag_clear
Function prototype	void qspi_flag_clear(qspi_type* qspi_x, uint32_t flag);
Function description	Clear flag
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	flag: to-be-cleared flag
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

qspi_flag

Clear the QSPI status flag

QSPI_CMDSTS_FLAG: QSPI command complete flag

Example:

```
qspi_flag_clear(QSPI_CMDSTS_FLAG);
```

5.18.9 qspi_dma_rx_threshold_set function

The table below describes the function qspi_dma_rx_threshold_set.

Table 509. qspi_dma_rx_threshold_set function

Name	Description
Function name	qspi_dma_rx_threshold_set
Function prototype	void qspi_dma_rx_threshold_set(qspi_type* qspi_x, qspi_dma_fifo_thod_type new_threshold);
Function description	Set QSPI DMA receive FIFO threshold
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	new_threshold: threshold
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

qspi_dma_fifo_thod_type

Configure a QSPI DMA FIFO threshold

QSPI_DMA_FIFO_THOD_WORD08: QSPI DMA FIFO 8 WORDs

QSPI_DMA_FIFO_THOD_WORD16: QSPI DMA FIFO 16 WORDs

QSPI_DMA_FIFO_THOD_WORD32: QSPI DMA FIFO 32 WORDs

Example:

```
qspi_dma_rx_threshold_set(QSPI_DMA_FIFO_THOD_WORD08);
```

5.18.10 qspi_dma_tx_threshold_set function

The table below describes the function qspi_dma_tx_threshold_set.

Table 510. qspi_dma_tx_threshold_set function

Name	Description
Function name	qspi_dma_tx_threshold_set
Function prototype	void qspi_dma_tx_threshold_set(qspi_type* qspi_x, qspi_dma_fifo_thod_type new_threshold);
Function description	Configure QSPI DMA transmit FIFO threshold
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	new_threshold: threshold Refer to *qspi_dma_fifo_thod_type*.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
qspi_dma_tx_threshold_set(QSPI_DMA_FIFO_THOD_WORD08);
```

5.18.11 qspi_dma_enable function

The table below describes the function qspi_dma_enable.

Table 511. qspi_dma_enable function

Name	Description
Function name	qspi_dma_enable
Function prototype	void qspi_dma_enable(qspi_type* qspi_x, confirm_state new_state);
Function description	Enable QSPI interrupts
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	new_state: DMA status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
qspi_dma_enable(QSPI1, TRUE);
```

5.18.12 qspi_busy_config function

The table below describes the function qspi_busy_config.

Table 512. qspi_busy_config function

Name	Description
Function name	qspi_busy_config
Function prototype	void qspi_busy_config(qspi_type* qspi_x, qspi_busy_pos_type busy_pos);
Function description	Configure the offset of QSPI busy bit in the status register
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	busy_pos: offset
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

qspi_busy_pos_type

QSPI_BUSY_OFFSET_0: Busy bit offset 0
QSPI_BUSY_OFFSET_1: Busy bit offset 1
QSPI_BUSY_OFFSET_2: Busy bit offset 2
QSPI_BUSY_OFFSET_3: Busy bit offset 3
QSPI_BUSY_OFFSET_4: Busy bit offset 4
QSPI_BUSY_OFFSET_5: Busy bit offset 5
QSPI_BUSY_OFFSET_6: Busy bit offset 6
QSPI_BUSY_OFFSET_7: Busy bit offset 7

Example:

```
qspi_busy_config(QSPI1, QSPI_BUSY_OFFSET_0);
```

5.18.13 qspi_xip_enable function

The table below describes the function qspi_xip_enable.

Table 513. qspi_xip_enable function

Name	Description
Function name	qspi_xip_enable
Function prototype	void qspi_xip_enable(qspi_type* qspi_x, confirm_state new_state);
Function description	Enable QSPI XIP port
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	new_state: XIP port status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
qspi_xip_enable(QSPI1, TRUE);
```

5.18.14 qspi_cmd_operation_kick function

The table below describes the function qspi_cmd_operation_kick.

Table 514. qspi_cmd_operation_kick function

Name	Description
Function name	qspi_cmd_operation_kick
Function prototype	void qspi_cmd_operation_kick(qspi_type* qspi_x, qspi_cmd_type* qspi_cmd_struct);
Function description	Start QSPI command operation
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	qspi_cmd_struct: command structure pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

qspi_cmd_type structure

The qspi_cmd_type is defined in the at32f435_437_qspi.h.

```
typedef struct
{
  confirm_state                pe_mode_enable;
  uint8_t                      pe_mode_operate_code;
  uint8_t                      instruction_code;
  qspi_cmd_inslen_type         instruction_length;
  uint32_t                     address_code;
  qspi_cmd_adrlen_type         address_length;
  uint32_t                     data_counter;
  uint8_t                      second_dummy_cycle_num;
  qspi_operate_mode_type       operation_mode;
  qspi_read_status_conf_type   read_status_config;
  confirm_state                read_status_enable;
  confirm_state                write_data_enable;
} qspi_cmd_type;
```

pe_mode_enable

Enable performance enhance mode according to the selected QSPI peripheral.

TRUE: Performance enhance mode enabled

FALSE: Performance enhance mode disabled

pe_mode_operate_code

Performance enhance mode operate code, dependent on the selected QSPI peripheral.

instruction_code

Command instruction code

instruction_length

Command instruction code length

QSPI_CMD_INSLEN_0_BYTE: No instruction code

QSPI_CMD_INSLEN_1_BYTE: 1-byte instruction code

QSPI_CMD_INSLEN_2_BYTE: 2-byte instruction code

address_code

Address code

address_length

Address length

QSPI_CMD_ADRLEN_0_BYTE: No address

QSPI_CMD_ADRLEN_1_BYTE: 1-byte address

QSPI_CMD_ADRLEN_2_BYTE: 2-byte address

QSPI_CMD_ADRLEN_3_BYTE: 3-byte address

QSPI_CMD_ADRLEN_4_BYTE: 4-byte address

data_counter

Number of data

second_dummy_cycle_num

Number of the second dummy cycle, ranging from 0 to 32

operation_mode

Operation mode

QSPI_OPERATE_MODE_111: qspi serial mode

QSPI_OPERATE_MODE_112: qspi dual mode

QSPI_OPERATE_MODE_114: qspi quad mode

QSPI_OPERATE_MODE_122: qspi dual i/o mode

QSPI_OPERATE_MODE_144: qspi quad i/o mode

QSPI_OPERATE_MODE_222: qspi instruction 2-bit mode

QSPI_OPERATE_MODE_444: qspi instruction 4-bit mode(qpi)

read_status_config

QSPI_RSTSC_HW_AUTO: Read by hardware automatically

QSPI_RSTSC_SW_ONCE: Read by software for once

read_status_enable

Enable read status

TRUE: Enable

FALSE: Disbale

write_data_enable

Enable write data

TRUE: Enable

FALSE: Disable

Example:

```
esmt32m_cmd_erase_config(&esmt32m_cmd_config, sec_addr);
qspi_cmd_operation_kick(QSPI1, &esmt32m_cmd_config);
```

5.18.15 qspi_xip_init function

The table below describes the function qspi_xip_init.

Table 515. qspi_xip_init function

Name	Description
Function name	qspi_xip_init
Function prototype	void qspi_xip_init(qspi_type* qspi_x, qspi_xip_type* xip_init_struct);
Function description	Initialize QSPI XIP port
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	xip_init_struct: initialization structure pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

qspi_xip_type structure

The qspi_xip_type is defined in the at32f435_437_qspi.h.

```
typedef struct
{
  uint8_t                    read_instruction_code;
  qspi_xip_addrlen_type      read_address_length;
  qspi_operate_mode_type     read_operation_mode;
  uint8_t                    read_second_dummy_cycle_num;
  uint8_t                    write_instruction_code;
  qspi_xip_addrlen_type      write_address_length;
  qspi_operate_mode_type     write_operation_mode;
  uint8_t                    write_second_dummy_cycle_num;
  qspi_xip_write_sel_type    write_select_mode;
  uint8_t                    write_time_counter;
  uint8_t                    write_data_counter;
  qspi_xip_read_sel_type     read_select_mode;
  uint8_t                    read_time_counter;
  uint8_t                    read_data_counter;
} qspi_xip_type;
```

read_instruction_code

Read command instruction code

read_address_length

Read command address length

QSPI_XIP_ADDRLEN_3_BYTE: 3-byte address

QSPI_XIP_ADDRLEN_4_BYTE: 4-byte address

read_operation_mode

Read command opration mdoe

QSPI_OPERATE_MODE_111: qspi serial mode

QSPI_OPERATE_MODE_112: qspi dual mode

QSPI_OPERATE_MODE_114: qspi quad mode

QSPI_OPERATE_MODE_122: qspi dual i/o mode

QSPI_OPERATE_MODE_144: qspi quad i/o mode
QSPI_OPERATE_MODE_222: qspi instruction 2-bit mode
QSPI_OPERATE_MODE_444: qspi instruction 4-bit mode(qpi)

read_second_dummy_cycle_num

Number of read command second dummy cycle, ranging from 0 to 32

write_instruction_code

Write command instruction code

write_address_length

Write command address length
QSPI_XIP_ADDRLEN_3_BYTE: 3-byte address
QSPI_XIP_ADDRLEN_4_BYTE: 4-byte address

write_operation_mode

Write command operation mode
QSPI_OPERATE_MODE_111: qspi serial mode
QSPI_OPERATE_MODE_112: qspi dual mode
QSPI_OPERATE_MODE_114: qspi quad mode
QSPI_OPERATE_MODE_122: qspi dual i/o mode
QSPI_OPERATE_MODE_144: qspi quad i/o mode
QSPI_OPERATE_MODE_222: qspi instruction 2-bit mode
QSPI_OPERATE_MODE_444: qspi instruction 4-bit mode(qpi)

write_second_dummy_cycle_num

Number of write command second dummy cycle, ranging from 0 to 32

write_select_mode

Write command mode selection
QSPI_XIPW_SEL_MODED: Mode D
QSPI_XIPW_SEL_MODET: Mode T

write_time_counter

Number of clock in write command mode T

write_data_counter

Number of data in write command mode D

read_select_mode

Read command mode selection
QSPI_XIPW_SEL_MODED: Mode D
QSPI_XIPW_SEL_MODET: Mode T

read_time_counter

Number of clock in read command mode T

read_data_counter

Number of data in read command mode T

Example:

```
/* initial xip */
xip_init_ly68l6400_config(&ly68l6400_xip_init);
qspi_xip_init(QSPI1, &ly68l6400_xip_init);
```

5.18.16 qspi_byte_read function

The table below describes the function qspi_byte_read.

Table 516. qspi_byte_read function

Name	Description
Function name	qspi_byte_read
Function prototype	uint8_t qspi_byte_read(qspi_type* qspi_x);
Function description	Read data in bytes
Input parameter	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Output parameter	NA
Return value	Return the data value.
Required preconditions	NA
Called functions	NA

Example:

```
uint8_t data;
data = qspi_byte_read(QSPI1);
```

5.18.17 qspi_half_word_read function

The table below describes the function qspi_half_word_read.

Table 517. qspi_half_word_read function

Name	Description
Function name	qspi_half_word_read
Function prototype	uint16_t qspi_half_word_read(qspi_type* qspi_x);
Function description	Read data in half-words.
Input parameter	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Output parameter	NA
Return value	Return the data value.
Required preconditions	NA
Called functions	NA

Example:

```
uint16_t data;
data = qspi_half_word_read(QSPI1);
```

5.18.18 qspi_word_read function

The table below describes the function qspi_word_read

Table 518. qspi_word_read function

Name	Description
Function name	qspi_word_read
Function prototype	uint32_t qspi_word_read(qspi_type* qspi_x);
Function description	Read data in words.
Input parameter	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Output parameter	NA
Return value	Return the data value.
Required preconditions	NA
Called functions	NA

Example:

```
uint32_t data;
data = qspi_word_read(QSPI1);
```

5.18.19 qspi_word_write function

The table below describes the function qspi_word_write.

Table 519. qspi_word_write function

Name	Description
Function name	qspi_word_write
Function prototype	void qspi_word_write(qspi_type* qspi_x, uint32_t value);
Function description	Write data in words.
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	value: the to-be-written data
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
qspi_word_write(QSPI1, 0x12345678);
```

5.18.20 qspi_half_word_write function

The table below describes the function qspi_half_word_write.

Table 520. qspi_half_word_write function

Name	Description
Function name	qspi_half_word_write
Function prototype	void qspi_half_word_write(qspi_type* qspi_x, uint16_t value);
Function description	Write data in half-words.
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	value: the to-be-written data
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
qspi_half_word_write(QSPI1, 0x1234);
```

5.18.21 qspi_byte_write function

The table below describes the function qspi_byte_write.

Table 521. qspi_byte_write function

Name	Description
Function name	qspi_byte_write
Function prototype	void qspi_byte_write(qspi_type* qspi_x, uint8_t value);
Function description	Write data in bytes.
Input parameter 1	qspi_x: selected QSPI peripheral This parameter can be QSPI1 or QSPI2.
Input parameter 2	value: the to-be-written data
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
qspi_byte_write(QSPI1, 0x12);
```

5.19 System configuration controller (SCFG)

The SCFG register structure scfg_type is defined in the “at32f435_437_scfg.h”.

```
/**
  * @brief type define scfg register all
  */
typedef struct
{
      …
} scfg_type;
```

The table below gives a list of the SCFG registers.

Table 522. 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 the SCFG library functions.

Table 523. Summary of SCFG library functions

Function name	Description
scfg_reset	SCFG reset
scfg_xmc_mapping_swap_set	Configure XMC address mapping swap
scfg_infrared_config	Infrared configuration
scfg_mem_map_set	Configure memory address mapping
scfg_emac_interface_set	Configure EMAC interface
scfg_exint_line_config	Configure external interrupt line
scfg_pins_ultra_driven_enable	Pin ultra-high current sinking capability enable

5.19.1 scfg_reset function

The table below describes the function scfg_reset.

Table 524. 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

Name	Description
Required preconditions	NA
Called functions	NA

Example:

```
scfg_reset();
```

5.19.2 scfg_xmc_mapping_swap_set function

The table below describes the function scfg_xmc_mapping_swap_set.

Table 525. scfg_xmc_mapping_swap_set function

Name	Description
Function name	scfg_xmc_mapping_swap_set
Function prototype	void scfg_xmc_mapping_swap_set(scfg_xmc_swap_type xmc_swap);
Function description	Configure XMC address mapping swap
Input parameter	xmc_swap: mapping swap parameter
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

scfg_xmc_swap_type

SCFG_XMC_SWAP_NONE: No XMC address mapping swap
SCFG_XMC_SWAP_MODE1: XMC address mapping swap mode 1
SCFG_XMC_SWAP_MODE2: XMC address mapping swap mode 2
SCFG_XMC_SWAP_MODE3: XMC address mapping swap mode 3

Example:

```
scfg_xmc_mapping_swap_set(SCFG_XMC_SWAP_MODE1);
```

5.19.3 scfg_infrared_config function

The table below describes the function scfg_infrared_config.

Table 526. 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 envelope signal source selection
Input parameter 2	polarity: Infrared output polarity selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

scfg_ir_source_type

Infrared modulation envelope signal source selection
SCFG_IR_SOURCE_TMR10: TMR10

scfg_ir_polarity_type

Infrared output polarity selection

SCFG_IR_POLARITY_NO_AFFECTE: Infrared output is not inversed

SCFG_IR_POLARITY_REVERSE: Infrared output is inversed

Example:

```
scfg_infrared_config(SCFG_IR_SOURCE_TMR10, SCFG_IR_POLARITY_NO_AFFECTE);
```

5.19.4 scfg_mem_map_set function

The table below describes the function scfg_mem_map_set.

Table 527. scfg_mem_map_set function

Name	Description
Function name	scfg_mem_map_set
Function prototype	void scfg_mem_map_set(scfg_mem_map_type mem_map);
Function description	Configure memory mapping at 0x00000000
Input parameter	mem_map: memory address mapping selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

scfg_mem_map_type

Memory address mapping type

SCFG_MEM_MAP_MAIN_MEMORY: Main Flash memory mapped at 0x00000000

SCFG_MEM_MAP_BOOT_MEMORY: Bootloader mapped at 0x00000000

SCFG_MEM_MAP_XMC_BANK1: XMC bank1 mapped at 0x00000000

SCFG_MEM_MAP_INTERNAL_SRAM: Embedded SRAM mapped at 0x00000000

SCFG_MEM_MAP_XMC_SDRAM_BANK1: XMC SDRAM bank1 mapped at 0x00000000

Example:

```
scfg_mem_map_set(SCFG_MEM_MAP_BOOT_MEMORY);
```

5.19.5 scfg_emac_interface_set function

The table below describes the function scfg_emac_interface_set.

Table 528. scfg_emac_interface_set function

Name	Description
Function name	scfg_emac_interface_set
Function prototype	void scfg_emac_interface_set(scfg_emac_interface_type mode);
Function description	Configure EMAC interface
Input parameter	mode: EMAC interface type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

scfg_emac_interface_type

EMAC interface type

SCFG_EMAC_SELECT_MII: MII
SCFG_EMAC_SELECT_RMII: RMII

Example:

```
scfg_emac_interface_set(SCFG_EMAC_SELECT_MII);
```

5.19.6 scfg_exint_line_config function

The table below describes the function scfg_exint_line_config.

Table 529. 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, scfg_pins_source_type pin_source);
Function description	Configure externl interrupt line
Input parameter 1	port_source: port source selection
Input parameter 2	pin_source: pin source selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

scfg_port_source_type

SCFG_PORT_SOURCE_GPIOA: GPIOA
SCFG_PORT_SOURCE_GPIOB: GPIOB
SCFG_PORT_SOURCE_GPIOC: GPIOC
SCFG_PORT_SOURCE_GPIOD: GPIOD
SCFG_PORT_SOURCE_GPIOE: GPIOE
SCFG_PORT_SOURCE_GPIOF: GPIOF
SCFG_PORT_SOURCE_GPIOG: GPIOG
SCFG_PORT_SOURCE_GPIOH: GPIOH

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: pin 15

Example:

```
scfg_exint_line_config(SCFG_PORT_SOURCE_GPIOA, SCFG_PINS_SOURCE1);
```

5.19.7 scfg_pins_ultra_driven_enable function

The table below describes the function scfg_pins_ultra_driven_enable.

Table 530. 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 selection
Input parameter 2	new_state: interrupt status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

scfg_ultra_driven_pins_type

SCFG_ULTRA_DRIVEN_PB3: PB3
SCFG_ULTRA_DRIVEN_PB9: PB9
SCFG_ULTRA_DRIVEN_PB10: PB10
SCFG_ULTRA_DRIVEN_PD12: PD12
SCFG_ULTRA_DRIVEN_PD13: PD13
SCFG_ULTRA_DRIVEN_PD14: PD14
SCFG_ULTRA_DRIVEN_PD15: PD15
SCFG_ULTRA_DRIVEN_PF14: PF14
SCFG_ULTRA_DRIVEN_PF15: PF15

Example:

```
scfg_pins_ultra_driven_enable(SCFG_ULTRA_DRIVEN_PB3, TRUE);
```

5.20 SDIO interface (SDIO)

The SDIO register structure crm_type is defined in the “at32f435_437_sdio.h”.

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

} sdio_type;
```

The table below gives a list of the SDIO registers.

Table 531. Summary of SDIO registers

Register	Description
pwrctrl	Power control register
clkctrl	Clock control register
arg	Argument register
cmd	Command register
rspcmd	Command response register
rsp1	Response register 1
rsp2	Response register 2
rsp3	Response register 3
rsp4	Response register 4
dttmr	Data timer register
dtlen	Data length register
dtctrl	Data control register
dtcntr	Data counter register
sts	Status register
intclr	Clear interrupt register
inten	Interrupt mask register
bufcntr	BUF counter register
buf	Data BUF register

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

Table 532. Summary of SDIO library functions

Function name	Description
sdio_reset	Reset SDIO peripheral registers and control status
sdio_power_set	Configure controller power status
sdio_power_status_get	Get controller power status
sdio_clock_config	Configure clock parameters
sdio_bus_width_config	Configure bus width
sdio_clock_bypass	Enable clock bypass mode
sdio_power_saving_mode_enable	Enable controller power-saving mode

sdio_flow_control_enable	Enable flow control mode
sdio_clock_enable	Enable clock
sdio_dma_enable	Enable DMA
sdio_interrupt_enable	Enable interrupts
sdio_flag_get	Get the flag
sdio_interrupt_flag_get	Get SDIO interrupt flag status
sdio_flag_clear	Clear the flag
sdio_command_config	Configure command argument
sdio_command_state_machine_enable	Enable command state machine
sdio_command_response_get	Get response command index
sdio_response_get	Get card command response
sdio_data_config	Configure data paramters
sdio_data_state_machine_enable	Enable data state machine
sdio_data_counter_get	Get the counter of to-be-sent data
sdio_data_read	Read one-WORD data from the receive FIFO
sdio_buffer_counter_get	Get the counter of data to be written into BUF or read from BUF
sdio_data_write	Write one-WORD data to the transmit FIFO
sdio_read_wait_mode_set	Configure read wait mode
sdio_read_wait_start	Read wait start
sdio_read_wait_stop	Read wait stop
sdio_io_function_enable	Enable IO function mode
sdio_io_suspend_command_set	Enable suspend command in IO function mode

5.20.1 sdio_reset function

The table below describes the function sdio_reset.

Table 533. sdio_reset function

Name	Description
Function name	sdio_reset
Function prototype	void sdio_reset(sdio_type *sdio_x);
Function description	Reset SDIO peripheral registers and control status
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* reset sdio */
sdio_reset(SDIO1);
```

5.20.2 sdio_power_set function

The table below describes the function sdio_power_set.

Table 534. sdio_power_set function

Name	Description
Function name	sdio_power_set
Function prototype	void sdio_power_set(sdio_type *sdio_x, sdio_power_state_type power_state);
Function description	Configure the controller power status
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	power_state: controller power status
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

power_state

Controller power status

SDIO_POWER_ON: Power ON

SDIO_POWER_OFF: Power OFF

Example:

```
/* sdio power on */
sdio_power_set(SDIO1, SDIO_POWER_ON);
```

5.20.3 sdio_power_status_get function

The table below describes the function sdio_power_status_get.

Table 535. sdio_power_status_get function

Name	Description
Function name	sdio_power_status_get
Function prototype	sdio_power_state_type sdio_power_status_get(sdio_type *sdio_x);
Function description	Get the controller power status
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	sdio_power_state_type: controller power status
Required preconditions	NA
Called functions	NA

Example:

```
/* check power status */
if(sdio_power_status_get(SDIO1) == SDIO_POWER_OFF)
{
  return SD_REQ_NOT_APPLICABLE;
}
```

5.20.4 sdio_clock_config function

The table below describes the function sdio_clock_config.

Table 536. sdio_clock_config function

Name	Description
Function name	sdio_clock_config
Function prototype	void sdio_clock_config(sdio_type *sdio_x, uint16_t clk_div, sdio_edge_phase_type clk_edg);
Function description	Configure clock parameters
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	clk_div: clock division, ranging from 0 to 0x3FF
Input parameter 2	clk_edg: clock edge configuration
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

clk_edg

Clock edge selection

SDIO_CLOCK_EDGE_RISING: Clock rising edge

SDIO_CLOCK_EDGE_FALLING: Clock falling edge

Example:

```
/* config sdio clock divide and edge phase */
sdio_clock_config(SDIO1, 0x2, SDIO_CLOCK_EDGE_FALLING);
```

5.20.5 sdio_bus_width_config function

The table below describes the function sdio_bus_width_config.

Table 537. sdio_bus_width_config function

Name	Description
Function name	sdio_bus_width_config
Function prototype	void sdio_bus_width_config(sdio_type *sdio_x, sdio_bus_width_type width);
Function description	Configure bus width
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	width: selected bus width
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

width

Data bus width selection

SDIO_BUS_WIDTH_D1: 1-bit data bus width

SDIO_BUS_WIDTH_D4: 4-bit data bus width

SDIO_BUS_WIDTH_D8: 8-bit data bus width

Example:

```
/* config sdio bus width */
sdio_bus_width_config(SDIOx, SDIO_BUS_WIDTH_D1);
```

5.20.6 sdio_clock_bypass function

The table below describes the function sdio_clock_bypass.

Table 538. sdio_clock_bypass function

Name	Description
Function name	sdio_clock_bypass
Function prototype	void sdio_clock_bypass(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable clock bypass mode
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state; enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* disable clock bypass */
sdio_clock_bypass(SDIO1, FALSE);
```

5.20.7 sdio_power_saving_mode_enable function

The table below describes the function sdio_power_saving_mode_enable.

Table 539. sdio_power_saving_mode_enable function

Name	Description
Function name	sdio_power_saving_mode_enable
Function prototype	void sdio_power_saving_mode_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable controller power-saving mode
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* disable power saving mode */
sdio_power_saving_mode_enable(SDIO1, FALSE);
```

5.20.8 sdio_flow_control_enable function

The table below describes the function sdio_flow_control_enable.

Table 540. sdio_flow_control_enable function

Name	Description
Function name	sdio_flow_control_enable
Function prototype	void sdio_flow_control_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable flow control mode
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* disable flow control */
sdio_flow_control_enable(SDIO1, FALSE);
```

5.20.9 sdio_clock_enable function

The table below describes the function sdio_clock_enable.

Table 541. sdio_clock_enable function

Name	Description
Function name	sdio_clock_enable
Function prototype	void sdio_clock_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable clock
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable to output sdio_ck */
sdio_clock_enable(SDIO1, TRUE);
```

5.20.10 sdio_dma_enable function

The table below describes the function sdio_dma_enable.

Table 542. sdio_dma_enable function

Name	Description
Function name	sdio_dma_enable
Function prototype	void sdio_dma_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable DMA
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable sdio dma */
sdio_dma_enable(SDIO1, TRUE);
```

5.20.11 sdio_interrupt_enable function

The table below describes the function sdio_interrupt_enable.

Table 543. crm_flag_clear function

Name	Description
Function name	sdio_interrupt_enable
Function prototype	void sdio_interrupt_enable(sdio_type *sdio_x, uint32_t int_opt, confirm_state new_state);
Function description	Enable interrupts
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	int_opt: selected interrupt type
Input parameter 3	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

int_opt

Interrupt type selection

SDIO_CMDFAIL_INT: Command CRC fail interrupt
SDIO_DTFAIL_INT: Data CRC fail interrupt
SDIO_CMDTIMEOUT_INT: Command timeout interrupt
SDIO_DTTIMEOUT_INT: Data timeout interrupt
SDIO_TXERRU_INT: TxBUF underrun error interrupt
SDIO_RXERRO_INT: RxBUF overrun error interrupt
SDIO_CMDRSPCMPL_INT: Command response received interrupt
SDIO_CMDCMPL_INT: Command sent interrupt
SDIO_DTCMP_INT: Data end interrupt

SDIO_SBITERR_INT: Start bit error interrupt
SDIO_DTBLKCMPL_INT: Data block end interrupt
SDIO_DOCMD_INT: Command acting interrupt
SDIO_DOTX_INT: Data transmit acting interrupt
SDIO_DORX_INT: Data receive acting interrupt
SDIO_TXBUFH_INT: TxBUF half empty interrupt
SDIO_RXBUFH_INT: RxBUF half empty interrupt
SDIO_TXBUFF_INT: TxBUF full interrupt
SDIO_RXBUFF_INT: RxBUF full interrupt
SDIO_TXBUFE_INT: TxBUF empty interrupt
SDIO_RXBUFE_INT: RxBUF empty interrupt
SDIO_TXBUF_INT: Data available in TxBUF interrupt
SDIO_RXBUF_INT: Data available in RxBUF interrupt
SDIO_SDIOIF_INT: SD I/O mode received interrupt

Example:

```
/* disable interrupt */
sdio_interrupt_enable(SDIO1, (SDIO_DTFAIL_INT   | SDIO_DTTIMEOUT_INT | \
                              SDIO_DTCMP_INT | SDIO_TXBUFH_INT | SDIO_RXBUFH_INT      | \
                              SDIO_TXERRU_INT| SDIO_RXERRO_INT | SDIO_SBITERR_INT), FALSE);
```

5.20.12 sdio_flag_get function

The table below describes the function sdio_flag_get.

Table 544. sdio_flag_get function

Name	Description
Function name	sdio_flag_get
Function prototype	flag_status sdio_flag_get(sdio_type *sdio_x, uint32_t flag);
Function description	Get the flag
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	flag: selected interrupt type
Output parameter	NA
Return value	flag_status: flag status This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

flag

Flag selection

SDIO_CMDFAIL_FLAG: Command CRC fail flag
SDIO_DTFAIL_FLAG: Data CRC fail flag
SDIO_CMDTIMEOUT_FLAG: Command timeout flag
SDIO_DTTIMEOUT_FLAG: Data timeout flag
SDIO_TXERRU_FLAG: TxBUF underrun error flag
SDIO_RXERRO_FLAG: RxBUF overrun error flag
SDIO_CMDRSPCMPL_FLAG: Command response received flag
SDIO_CMDCMPL_FLAG: Command sent flag
SDIO_DTCMP_FLAG: Data transfer complete flag

SDIO_SBITERR_FLAG: Start bit error flag
SDIO_DTBLKCMPL_FLAG: Data block transfer complete flag
SDIO_DOCMD_FLAG: Command acting flag
SDIO_DOTX_FLAG: Data transmit acting flag
SDIO_DORX_FLAG: Data receive acting flag
SDIO_TXBUFH_FLAG: TxBUF half-empty flag
SDIO_RXBUFH_FLAG: RxBUF half-empty flag
SDIO_TXBUFF_FLAG: TxBUF full flag
SDIO_RXBUFF_FLAG: RxBUF full flag
SDIO_TXBUFE_FLAG: TxBUF empty flag
SDIO_RXBUFE_FLAG: RxBUF empty flag
SDIO_TXBUF_FLAG: Data available in TxBUF flag
SDIO_RXBUF_FLAG: Data available in RxBUF flag
SDIO_SDIOIF_FLAG: SD I/O mode received flag

Example:

```
/* check dttimeout flag */
if(sdio_flag_get(SDIOx, SDIO_DTTIMEOUT_FLAG) != RESET)
{
}
```

5.20.13 sdio_interrupt_flag_get function

The table below describes the function sdio_interrupt_flag_get.

Table 545. sdio_interrupt_flag_get function

Name	Description
Function name	sdio_interrupt_flag_get
Function prototype	flag_status sdio_interrupt_flag_get(sdio_type *sdio_x, uint32_t flag);
Function description	Get SDIO interrupt flag status
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	flag: selected an interrupt flag
Output parameter	NA
Return value	flag_status: SET or RESET
Required preconditions	NA
Called functions	NA

flag

Flag selection

SDIO_CMDFAIL_FLAG: Command CRC fail flag
SDIO_DTFAIL_FLAG: Data CRC fail flag
SDIO_CMDTIMEOUT_FLAG: Command timeout flag
SDIO_DTTIMEOUT_FLAG: Data timeout flag
SDIO_TXERRU_FLAG: TxBUF underrun error flag
SDIO_RXERRO_FLAG: RxBUF overrun error flag
SDIO_CMDRSPCMPL_FLAG: Command response received flag
SDIO_CMDCMPL_FLAG: Command sent flag
SDIO_DTCMP_FLAG: Data transfer complete flag
SDIO_SBITERR_FLAG: Start bit error flag

SDIO_DTBLKCMPL_FLAG: Data block transfer complete flag
SDIO_DOCMD_FLAG: Command acting flag
SDIO_DOTX_FLAG: Data transmit acting flag
SDIO_DORX_FLAG: Data receive acting flag
SDIO_TXBUFH_FLAG: TxBUF half-empty flag
SDIO_RXBUFH_FLAG: RxBUF half-empty flag
SDIO_TXBUFF_FLAG: TxBUF full flag
SDIO_RXBUFF_FLAG: RxBUF full flag
SDIO_TXBUFE_FLAG: TxBUF empty flag
SDIO_RXBUFE_FLAG: RxBUF empty flag
SDIO_TXBUF_FLAG: Data available in TxBUF flag
SDIO_RXBUF_FLAG: Data available in RxBUF flag
SDIO_SDIOIF_FLAG: SD I/O mode received flag

Example:

```
/* check dttimeout interrupt flag */
if(sdio_interrupt_flag_get(SDIOx, SDIO_DTTIMEOUT_FLAG) != RESET)
{
}
```

5.20.14 sdio_flag_clear function

The table below describes the function sdio_flag_clear.

Table 546. sdio_flag_clear function

Name	Description
Function name	sdio_flag_clear
Function prototype	void sdio_flag_clear(sdio_type *sdio_x, uint32_t flag);
Function description	Clear flag
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	flag: selected interrupt type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flag

Flag selection

SDIO_CMDFAIL_FLAG: Command CRC fail flag
SDIO_DTFAIL_FLAG: Data CRC fail flag
SDIO_CMDTIMEOUT_FLAG: Command timeout flag
SDIO_DTTIMEOUT_FLAG: Data timeout flag
SDIO_TXERRU_FLAG: TxBUF underrun error flag
SDIO_RXERRO_FLAG: RxBUF overrun error flag
SDIO_CMDRSPCMPL_FLAG: Command response received flag
SDIO_CMDCMPL_FLAG: Command transfer complete flag
SDIO_DTCMP_FLAG: Data transfer complete flag
SDIO_SBITERR_FLAG: Start bit error flag
SDIO_DTBLKCMPL_FLAG: Data block transfer complete flag

SDIO_SDIOIF_FLAG: SD I/O mode received flag

Example:

```
/* clear flags */
#define SDIO_STATIC_FLAGS ((uint32_t)0x000005FF)
sdio_flag_clear(SDIO1, SDIO_STATIC_FLAGS);
```

5.20.15 sdio_command_config function

The table below describes the function sdio_command_config.

Table 547. sdio_command_config function

Name	Description
Function name	sdio_command_config
Function prototype	void sdio_command_config(sdio_type *sdio_x, sdio_command_struct_type *command_struct);
Function description	Configure command argument
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	command_struct: sdio_command_struct_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

command_struct

The sdio_command_struct_type is defined in the at32f435_437_sdio.h.

```
typedef struct
{
  uint32_t                argument;
  uint8_t                 cmd_index;
  sdio_reponse_type       rsp_type;
  sdio_wait_type          wait_type;
} sdio_command_struct_type;
```

argument

Command argument is sent to a card as part of a command. It is dependent on the command type.

cmd_index

Command index

rsp_type

Response type, dependent on the command type, including:

SDIO_RESPONSE_NO: No response

SDIO_RESPONSE_SHORT: Short response

SDIO_RESPONSE_LONG: Long response

wait_type

Wait type, dependent on the command type, including:

SDIO_WAIT_FOR_NO: No wait

SDIO_WAIT_FOR_INT: Wait for interrupt request

SDIO_WAIT_FOR_PEND: Wait for end of transfer

Example:

```
/* send cmd16, set block length */
sdio_command_struct_type sdio_command_init_struct;
sdio_command_init_struct.argument = (uint32_t)8;
sdio_command_init_struct.cmd_index = SD_CMD_SET_BLOCKLEN;
sdio_command_init_struct.rsp_type = SDIO_RESPONSE_SHORT;
sdio_command_init_struct.wait_type = SDIO_WAIT_FOR_NO;
/* sdio command config */
sdio_command_config(SDIOx, &sdio_command_init_struct);
```

5.20.16 sdio_command_state_machine_enable function

The table below describes the function sdio_command_state_machine_enable.

Table 548. sdio_command_state_machine_enable function

Name	Description
Function name	sdio_command_state_machine_enable
Function prototype	void sdio_command_state_machine_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable command state machine
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable ccsm */
sdio_command_state_machine_enable(SDIO1, TRUE);
```

5.20.17 sdio_command_response_get function

The table below describes the function sdio_command_response_get.

Table 549. sdio_command_response_get function

Name	Description
Function name	sdio_command_response_get
Function prototype	uint8_t sdio_command_response_get(sdio_type *sdio_x);
Function description	Get response command index
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	uint8_t: response command index
Required preconditions	NA
Called functions	NA

Example:

```
/* get response of command index */
uint8_t rsp_cmd = 0;
rsp_cmd = sdio_command_response_get(SDIO1);
```

5.20.18 sdio_response_get function

The table below describes the function sdio_response_get

Table 550. sdio_response_get function

Name	Description
Function name	sdio_response_get
Function prototype	uint32_t sdio_response_get(sdio_type *sdio_x, sdio_rsp_index_type reg_index);
Function description	Get card command response
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	reg_index: response register number (1/2/3/4)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

reg_div

Response register selection

SDIO_RSP1_INDEX: Response register 1

SDIO_RSP2_INDEX: Response register 2

SDIO_RSP3_INDEX: Response register 3

SDIO_RSP4_INDEX: Response register 4

Example:

```
/* get response register1 */
response = sdio_response_get(SDIO1, SDIO_RSP1_INDEX);
```

5.20.19 sdio_data_config function

The table below describes the function sdio_data_config.

Table 551. sdio_data_config function

Name	Description
Function name	sdio_data_config
Function prototype	void sdio_data_config(sdio_type *sdio_x, sdio_data_struct_type *data_struct);
Function description	Configure data paremeters
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	data_struct: sdio_data_struct_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

data_struct

The sdio_data_struct_type is defined in the at32f435_437_sdio.h.

```
typedef struct
{
  uint32_t                    timeout;
  uint32_t                    data_length;
  sdio_block_size_type        block_size;
```

```
  sdio_transfer_mode_type          transfer_mode;
  sdio_transfer_direction_type     transfer_direction;
} sdio_data_struct_type;
```

timeout

Data transfer timeout, with the bus clock as the counting base

data_length

Length of the to-be-sent data

block_size

Block size, including

SDIO_DATA_BLOCK_SIZE_1B:	1-bit
SDIO_DATA_BLOCK_SIZE_2B:	2-bit
SDIO_DATA_BLOCK_SIZE_4B:	4-bit
SDIO_DATA_BLOCK_SIZE_8B:	8-bit
SDIO_DATA_BLOCK_SIZE_16B:	16-bit
SDIO_DATA_BLOCK_SIZE_32B:	32-bit
SDIO_DATA_BLOCK_SIZE_64B:	64-bit
SDIO_DATA_BLOCK_SIZE_128B:	128-bit
SDIO_DATA_BLOCK_SIZE_256B:	256-bit
SDIO_DATA_BLOCK_SIZE_512B:	512-bit
SDIO_DATA_BLOCK_SIZE_1024B:	1024-bit
SDIO_DATA_BLOCK_SIZE_2048B:	2048-bit
SDIO_DATA_BLOCK_SIZE_4096B:	4096-bit
SDIO_DATA_BLOCK_SIZE_8192B:	8192-bit
SDIO_DATA_BLOCK_SIZE_16384B:	16384-bit

transfer_mode

Data transfer mode selection

SDIO_DATA_BLOCK_TRANSFER:	Data block mode
SDIO_DATA_STREAM_TRANSFER:	Stream mode

transfer_direction

Data transfer direction selection

SDIO_DATA_TRANSFER_TO_CARD:	Controller-to-card
SDIO_DATA_TRANSFER_TO_CONTROLLER:	Card-to-controller

Example:

```
sdio_data_struct_type sdio_data_init_struct;
sdio_data_init_struct.block_size = SDIO_DATA_BLOCK_SIZE_512B;
sdio_data_init_struct.data_length = 8 ;
sdio_data_init_struct.timeout = SD_DATATIMEOUT ;
sdio_data_init_struct.transfer_direction = SDIO_DATA_TRANSFER_TO_CARD;
sdio_data_init_struct.transfer_mode = SDIO_DATA_BLOCK_TRANSFER;
/* config sdio data */
sdio_data_config(SDIO1, &sdio_data_init_struct);
```

5.20.20 sdio_data_state_machine_enable function

The table below describes the function sdio_data_state_machine_enable.

Table 552. sdio_data_state_machine_enable function

Name	Description
Function name	sdio_data_state_machine_enable
Function prototype	void sdio_data_state_machine_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable data state machine
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state; enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable dcsm */
sdio_data_state_machine_enable(SDIO1, TRUE);
```

5.20.21 sdio_data_counter_get function

The table below describes the function sdio_data_counter_get.

Table 553. sdio_data_counter_get function

Name	Description
Function name	sdio_data_counter_get
Function prototype	uint32_t sdio_data_counter_get(sdio_type *sdio_x);
Function description	Get the counter of to-be-sent data
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	uint32_t: the counter of to-be-sent data
Required preconditions	NA
Called functions	NA

Example:

```
/* get data counter */
uint32_t count = 0;
count = sdio_data_counter_get (SDIO1);
```

5.20.22 sdio_data_read function

The table below describes the function sdio_data_read.

Table 554. sdio_data_read function

Name	Description
Function name	sdio_data_read
Function prototype	uint32_t sdio_data_read(sdio_type *sdio_x);
Function description	Read one-WORD data from the receive FIFO
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	NA
Input parameter 3	NA
Output parameter	NA
Return value	uint32_t: one-WORD data
Required preconditions	NA
Called functions	NA

Example:

```
/* read data */
uint32_t data = 0;
data = sdio_data_read(SDIO1);
```

5.20.23 sdio_buffer_counter_get function

The table below describes the function sdio_buffer_counter_get.

Table 555. sdio_buffer_counter_get function

Name	Description
Function name	sdio_buffer_counter_get
Function prototype	uint32_t sdio_buffer_counter_get(sdio_type *sdio_x);
Function description	Get the counter of data to be written into BUF or read from BUF
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* get buffer count */
uint32_t count = 0;
count = sdio_buffer_counter_get(SDIO1);
```

5.20.24 sdio_data_write function

The table below describes the function sdio_data_write

Table 556. sdio_data_write function

Name	Description
Function name	sdio_data_write
Function prototype	void sdio_data_write(sdio_type *sdio_x, uint32_t data);
Function description	Write one-WORD data to the transmit FIFO
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* write data */
uin32_t data = 0x11223344;
sdio_data_write(SDIO1, data);
```

5.20.25 sdio_read_wait_mode_set function

The table below describes the function sdio_read_wait_mode_set.

Table 557. sdio_read_wait_mode_set function

Name	Description
Function name	sdio_read_wait_mode_set
Function prototype	void sdio_read_wait_mode_set(sdio_type *sdio_x, sdio_read_wait_mode_type mode);
Function description	Configure read wait mode
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	mode: read wait mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

mode

SDIO_READ_WAIT_CONTROLLED_BY_D2: Read wait is controlled by DATA Line2

SDIO_READ_WAIT_CONTROLLED_BY_CK: Read wait is controlled by clock line

Example:

```
/* config read wait mode */
sdio_read_wait_mode_set(SDIO1, SDIO_READ_WAIT_CONTROLLED_BY_D2);
```

5.20.26 sdio_read_wait_start function

The table below describes the function sdio_read_wait_start.

Table 558. sdio_read_wait_start function

Name	Description
Function name	sdio_read_wait_start
Function prototype	void sdio_read_wait_start(sdio_type *sdio_x, confirm_state new_state);
Function description	Read wait start
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Calling this function indicates start of read wait; when this function is called to disable wait state, it indicates that no action occurs.

Example:

```
/* start read wait mode */
sdio_read_wait_start (SDIO1, TRUE);
```

5.20.27 sdio_read_wait_stop function

The table below describes the function sdio_read_wait_stop.

Table 559. sdio_read_wait_stop function

Name	Description
Function name	sdio_read_wait_stop
Function prototype	void sdio_read_wait_stop(sdio_type *sdio_x, confirm_state new_state);
Function description	Read wait stop
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Calling this function indicates stop of read wait; when this function is called to disable wait state, it indicates that the read wait is in progress.

Example:

```
/* stop read wait mode */
sdio_read_wait_stop (SDIO1, TRUE);
```

5.20.28 sdio_io_function_enable function

The table below describes the function sdio_io_function_enable.

Table 560. sdio_io_function_enable function

Name	Description
Function name	sdio_io_function_enable
Function prototype	void sdio_io_function_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable IO function mode
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable sdio IO mode */
sdio_io_function_enable (SDIO1, TRUE);
```

5.20.29 sdio_io_suspend_command_set function

The table below describes the function sdio_io_suspend_command_set.

Table 561. sdio_io_suspend_command_set function

Name	Description
Function name	sdio_io_suspend_command_set
Function prototype	void sdio_io_suspend_command_set(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable suspend command in IO function mode
Input parameter 1	sdio_x: selected SDIO peripheral, such as SDIO1
Input parameter 2	new_state: new state, enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* send suspend command */
sdio_io_suspend_command_set (SDIO1, TRUE);
```

5.21 Serial peripheral interface (SPI)/I²S

The SPI register structure spi_type is defined in the “at32f435_437_spi.h”.

```
/**
  * @brief type define spi register all
  */
typedef struct
{
  ...
} spi_type;
```

The table below gives a list of the SPI registers.

Table 562. Summary of SPI registers

Registers	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 the SPI library functions.

Table 563. Summary of SPI library functions

Function name	Description
spi_i2s_reset	Reset SPI/I²S 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 I²S initialization structure
i2s_init	Initialize I²S
i2s_enable	Enable I²S

spi_i2s_interrupt_enable	Enable SPI/I²S interrupts
spi_i2s_dma_transmitter_enable	Enable SPI/I²S DMA transmit
spi_i2s_dma_receiver_enable	Enable SPI/I²S DMA receive
spi_i2s_data_transmit	SPI/I²S transmits data
spi_i2s_data_receive	SPI/I²S receives data
spi_i2s_flag_get	Get SPI/I²S flag
spi_i2s_flag_clear	Clear SPI/I²S flag

5.21.1 spi_i2s_reset function

The table below describes the function spi_i2s_reset.

Table 564. 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/I²S registers to their reset values
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4, I2S2EXT or I2S3EXT.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

Example:

```
spi_i2s_reset (SPI1);
```

5.21.2 spi_default_para_init function

The table below describes the function spi_default_para_init.

Table 565. 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: *spi_init_type* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of the spi_init_type before starting.
Called functions	NA

Example:

```
spi_init_type spi_init_struct;
spi_default_para_init (&spi_init_struct);
```

5.21.3 spi_init function

The table below describes the function spi_init.

Table 566. 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: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3 or SPI4.
Input parameter 2	spi_init_struct: *spi_init_type* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of the spi_init_type before starting.
Called functions	NA

The spi_init_type is defined in the at32f435_437_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_3: Divided by 3
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);
```

5.21.4 spi_ti_mode_enable function

The table below describes the function spi_ti_mode_enable.

Table 567. 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: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3 or SPI4.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* spi ti mode enable */
spi_ti_mode_enable (SPI1, TRUE);
```

5.21.5 spi_crc_next_transmit function

The table below describes the function spi_crc_next_transmit.

Table 568. 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: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3 or SPI4.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
spi_crc_next_transmit (SPI1);
```

5.21.6 spi_crc_polynomial_set function

The table below describes the function spi_crc_polynomial_set.

Table 569. 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: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3 or SPI4.
Input parameter 2	crc_poly: CRC polynomial Range: 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);
```

5.21.7 spi_crc_polynomial_get function

The table below describes the function spi_crc_polynomial_get.

Table 570. 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: selected SPI peripheral This parameter can be SPI1，SPI2，SPI3，SPI4.
Output parameter	NA
Return value	CRC polynomial Range: 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);
```

5.21.8 spi_crc_enable function

The table below describes the function spi_crc_enable.

Table 571. 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: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3 or SPI4.
Input parameter 2	new_state: enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* spi crc enable */
spi_crc_enable (SPI1, TRUE);
```

5.21.9 spi_crc_value_get function

The table below describes the function spi_crc_value_get.

Table 572. 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: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3 or SPI4.
Input parameter 2	*crc_direction*: 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);
```

5.21.10 spi_hardware_cs_output_enable function

The table below describes the function spi_hardware_cs_output_enable.

Table 573. 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: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3 or SPI4.
Input parameter 2	new_state: enable or disable This parameter can be 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);
```

5.21.11 spi_software_cs_internal_level_set function

The table below describes the function spi_software_cs_internal_level_set.

Table 574. 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: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3 or SPI4.
Input parameter 2	*level*: set software CS internal level
Output parameter	NA
Return value	NA
Required preconditions	1. This setting is applicable to software CS mode only 2. In master mode, the “level” value must be “SPI_SWCS_INTERNAL_LEVEL_HIGHT”.
Called functions	NA

level

Set software CS internal level

SPI_SWCS_INTERNAL_LEVEL_LOW: Software CS internal low level

SPI_SWCS_INTERNAL_LEVEL_HIGHT: Software CS internal high level

Example:

```
/* set the internal level high */
spi_software_cs_internal_level_set (SPI1, SPI_SWCS_INTERNAL_LEVEL_HIGHT);
```

5.21.12 spi_frame_bit_num_set function

The table below describes the function spi_frame_bit_num_set.

Table 575. spi_frame_bit_num_set function

Name	Description
Function name	spi_frame_bit_num_set
Function prototype	void spi_frame_bit_num_set(spi_type* spi_x, spi_frame_bit_num_type bit_num);
Function description	Set the number of bits in a frame
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3 or SPI4.
Input parameter 2	*bit_num*: 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);
```

5.21.13 spi_half_duplex_direction_set function

The table below describes the function spi_half_duplex_direction_set.

Table 576. 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: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3 or SPI4.
Input parameter 2	*direction*: 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);
```

5.21.14 spi_enable function

The table below describes the function spi_enable.

Table 577. 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: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3 or SPI4.
Input parameter 2	new_state: enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable spi */
spi_enable (SPI1, TRUE);
```

5.21.15 i2s_default_para_init function

The table below describes the function i2s_default_para_init.

Table 578. 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 I²S initialization structure
Input parameter 1	i2s_init_struct: *spi_i2s_flag* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of the i2s_init_type before starting.
Called functions	NA

Example:

```
i2s_init_type i2s_init_struct;
i2s_default_para_init (&i2s_init_struct);
```

5.21.16 i2s_init function

The table below describes the function i2s_init.

Table 579. 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 I²S
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4, I2S2EXT or I2S3EXT.
Input parameter 2	i2s_init_struct: *spi_i2s_flag* pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of the i2s_init_type before starting.
Called functions	NA

The i2s_init_type is defined in the at32f435_437_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

I²S transfer mdoe

I2S_MODE_SLAVE_TX: I2S slave transmit

I2S_MODE_SLAVE_RX: I2S slave receive

I2S_MODE_MASTER_TX: I2S mastr transmit

I2S_MODE_MASTER_RX: I2S master receive

audio_protocol

I²S 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

I²S audio sampling frequency

I2S_AUDIO_FREQUENCY_DEFAULT:

Kept at its reset value (sampling frequency changes with SCLK)

I2S_AUDIO_FREQUENCY_8K: I2S sampling frequency 8K

I2S_AUDIO_FREQUENCY_11_025K: I2S sampling frequency 11.025K

I2S_AUDIO_FREQUENCY_16K: I2S sampling frequency 16K
I2S_AUDIO_FREQUENCY_22_05K: I2S sampling frequency 22.05K
I2S_AUDIO_FREQUENCY_32K: I2S sampling frequency 32K
I2S_AUDIO_FREQUENCY_44_1K: I2S sampling frequency 44.1K
I2S_AUDIO_FREQUENCY_48K: I2S sampling frequency 48K
I2S_AUDIO_FREQUENCY_96K: I2S sampling frequency 96K
I2S_AUDIO_FREQUENCY_192K: I2S sampling frequency 192K

data_channel_format

I²S 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

I²S 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);
```

5.21.17 i2s_enable function

The table below describes the function i2s_enable.

Table 580. 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 I²S
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3 or SPI4.
Input parameter 2	new_state: enable or disable This parameter can be FALSE orTRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable i2s*/
i2s_enable (SPI1, TRUE);
```

5.21.18 spi_i2s_interrupt_enable function

The table below describes the function spi_i2s_interrupt_enable.

Table 581. 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/I²S interrupts
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4, I2S2EXT or I2S3EXT.
Input parameter 2	*spi_i2s_int*: SPI interrupt selection
Input parameter 3	new_state: enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

spi_i2s_int

SPI/I²S interrupt selection

SPI_I2S_ERROR_INT:

SPI/I²S 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);
```

5.21.19 spi_i2s_dma_transmitter_enable function

The table below describes the function spi_i2s_dma_transmitter_enable.

Table 582. 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/I²S DMA transmitter
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4, I2S2EXT or I2S3EXT.
Input parameter 2	new_state: enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable spi transmitter dma */
spi_i2s_dma_transmitter_enable (SPI1, TRUE);
```

5.21.20 spi_i2s_dma_receiver_enable function

The table below describes the function spi_i2s_dma_receiver_enable.

Table 583. spi_i2s_dma_receiver_enable function

Name	Description
Function name	spi_i2s_dma_receiver_enable
Function prototype	void spi_i2s_dma_receiver_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable SPI/I²S DMA receiver
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4, I2S2EXT or I2S3EXT.
Input parameter 2	new_state: enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable spi dma transmitter */
spi_i2s_dma_transmitter_enable (SPI1, TRUE);
```

5.21.21 spi_i2s_data_transmit function

The table below describes the function spi_i2s_data_transmit.

Table 584. spi_i2s_data_transmit function

Name	Description
Function name	spi_i2s_data_transmit
Function prototype	void spi_i2s_data_transmit(spi_type* spi_x, uint16_t tx_data);
Function description	SPI/I²S sends data
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4, I2S2EXT or I2S3EXT.
Input parameter 2	tx_data: data to be sent Value range (for 8-bit bit in a frame): 0x00~0xFF Value range (for 16-bit 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);
```

5.21.22 spi_i2s_data_receive function

The table below describes the function spi_i2s_data_receive.

Table 585. 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/I²S receives data
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4, I2S2EXT or I2S3EXT.
Output parameter	rx_data: data to receive Value range (for 8-bit bit in a frame): 0x00~0xFF Value range (for 16-bit 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);
```

5.21.23 spi_i2s_flag_get function

The table below describes the function spi_i2s_flag_get.

Table 586. 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/I²S flag
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4, I2S2EXT or I2S3EXT.
Input parameter 2	spi_i2s_flag: flag selection Refer to “spi_i2s_flag” description for details.
Output parameter	NA
Return value	flag_status: flag status This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

spi_i2s_flag

SPI/I²S is used to select a flag from the optional parameters below:

SPI_I2S_RDBF_FLAG: SPI/I²S receive data buffer full
SPI_I2S_TDBE_FLAG: SPI/I²S transmit data buffer empty
I2S_ACS_FLAG: I2S audio channel state (indicating left/right channel)
I2S_TUERR_FLAG: I2S transmitter underload error
SPI_CCERR_FLAG: SPI CRC error
SPI_MMERR_FLAG: SPI master mode error
SPI_I2S_ROERR_FLAG: SPI/I²S receive overflow error
SPI_I2S_BF_FLAG: SPI/I²S busy
SPI_CSPAS_FLAG: SPI CS pulse error

Example:

```
/* get receive data buffer full flag */
flag_status status;
status = spi_i2s_flag_get(SPI1, SPI_I2S_RDBF_FLAG);
```

5.21.24 spi_i2s_interrupt_flag_get function

The table below describes the function spi_i2s_flag_clear.

Table 587. spi_i2s_flag_clear function

Name	Description
Function name	spi_i2s_interrupt_flag_get
Function prototype	flag_status spi_i2s_interrupt_flag_get(spi_type* spi_x, uint32_t spi_i2s_flag);
Function description	Get SPI/I²S interrupt flag status
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4, I2S2EXT or I2S3EXT.
Input parameter 2	*spi_i2s_flag:* to-be-cleared flag Refer to “spi_i2s_flag” below for details.
Output parameter	NA
Return value	flag_status: SET or RESET.
Required preconditions	NA
Called functions	NA

spi_i2s_flag:

SPI/I²S is used for flag selection, including:

SPI_I2S_RDBF_FLAG: SPI/I²S receive data buffer full
SPI_I2S_TDBE_FLAG: SPI/I²S transmit data buffer empty
I2S_TUERR_FLAG: I2S transmitter underload error
SPI_CCERR_FLAG: SPI CRC error
SPI_MMERR_FLAG: SPI master mode error
SPI_I2S_ROERR_FLAG: SPI/I²S 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(SPI1, SPI_I2S_RDBF_FLAG);
```

5.21.25 spi_i2s_flag_clear function

The table below describes the function spi_i2s_flag_clear.

Table 588. 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/I²S flag
Input parameter 1	spi_x: selected SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4, I2S2EXT or I2S3EXT.
Input parameter 2	*spi_i2s_flag:* to-be-cleared flag Refer to “spi_i2s_flag” description for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

spi_i2s_flag:

SPI/I²S is used for flag selection, including:

SPI_I2S_RDBF_FLAG: SPI/I²S receive data buffer full

I2S_TUERR_FLAG: I2S transmitter underload error

SPI_CCERR_FLAG: SPI CRC error

SPI_MMERR_FLAG: SPI master mode error

SPI_I2S_ROERR_FLAG: SPI/I²S receive overflow error

SPI_CSPAS_FLAG: SPI CS pulse error

Note: The SPI_I2S_TDBE_FLAG (SPI/I2S transmit data buffer empty），I2S_ACS_FLAG (I2S audio channel state) and SPI_I2S_BF_FLAG (SPI/I2S busy) are all set and cleared by hardware to indicate communication state, without the intervention of software.

Example:

```
/* clear receive data buffer full flag */
spi_i2s_flag_clear (SPI1, SPI_I2S_RDBF_FLAG);
```

5.22 SysTick

The SysTick register structure SysTick_Type is defined in the “core_cm4.h”.

```
typedef struct
{
    ...

} SysTick_Type;
```

The table below gives a list of SysTick registers.

Table 589. Summary of SysTick registers

Register	Description
ctrl	Control 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 590. 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

5.22.1 systick_clock_source_config function

The table below describes the function systick_clock_source_config.

Table 591. 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);
```

5.22.2 SysTick_Config function

The table below describes the function SysTick_Config.

Table 592. 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
Input parameter 2	
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);
```

5.23 Timer (TMR)

The TMR register structure tmr_type is defined in the “at32f435_437_tmr.h”.

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


} tmr_type;
```

The table below gives a list of TMR registers.

Table 593. 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 register
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
cm3	TMR channel mode register 3
c5dt	TMR channel 5 data register

The table below gives a list of TMR library functions.

Table 594. Summary of TMR library functions

Function name	Description
tmr_reset	TMR is reset by CRM reset register
tmr_counter_enable	Enable or disable TMR counter
tmr_output_default_para_init	Initialize TMR output default parameter
tmr_input_default_para_init	Initialize TMR input default paramter
tmr_brkdt_default_para_init	Initialize TMR brkdt default paramter
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 value
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 channel
tmr_output_channel_mode_select	Select TMR output channel mode
tmr_period_value_set	Set TMR period value
tmr_period_value_get	Get TMR period value
tmr_channel_value_set	Set TMR channel value
tmr_channel_value_get	Get TMR channel value
tmr_period_buffer_enable	Enable or disable TMR periodic buffer
tmr_output_channel_buffer_enable	Enable or disable TMR output channel buffer
tmr_output_channel_immediately_set	TMR output channel enable immediately
tmr_output_channel_switch_set	Set TMR output channel switch
tmr_one_cycle_mode_enable	Enable or disable TMR one-cycle mode
tmr_32_bit_function_enable	Enable or disable TMR 32-bit function (plus mode)
tmr_overflow_request_source_set	Select TMR overflow event source
tmr_overflow_event_disable	Enable or disable TMR overflow event generation
tmr_input_channel_init	Initialize TMR input channel
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_trgout2_enable	Enable or disable TMR trigger output 2 signal
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 flag
tmr_flag_clear	Clear TMR flag
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 encoder 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

5.23.1 tmr_reset function

The table below describes the function tmr_reset.

Table 595. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

Example:

```
tmr_reset(TMR1);
```

5.23.2 tmr_counter_enable function

The table below describes the function tmr_counter_enable.

Table 596. 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 counter
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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);
```

5.23.3 tmr_output_default_para_init function

The table below describes the function tmr_output_default_para_init.

Table 597. 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 tmr_output_struct.

Table 598. tmr_output_struct default values

Member	Default value
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);
```

5.23.4 tmr_input_default_para_init function

The table below describes the function tmr_input_default_para_init.

Table 599. 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 tmr_input_struct.

Table 600. 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);
```

5.23.5 tmr_brkdt_default_para_init function

The table below describes the function tmr_brkdt_default_para_init.

Table 601. 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 tmr_brkdt_struct.

Table 602. tmr_brkdt_struct default values

Member	Default value
deadtime	0x0
brk_polarity	TMR_BRK_INPUT_ACTIVE_LOW
wp_level	TMR_WP_OFF

Member	Default value
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);
```

5.23.6 tmr_base_init function

The table below describes the function tmr_base_init.

Table 603. 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 frequency division
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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);
```

5.23.7 tmr_clock_source_div_set function

The table below describes the function tmr_clock_source_div_set.

Table 604. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	tmr_clock_div: TMR clock source division factor
Output parameter	NA
Return value	NA

Name	Description
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);
```

5.23.8 tmr_cnt_dir_set function

The table below describes the function tmr_cnt_dir_set.

Table 605. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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);
```

5.23.9 tmr_repetition_counter_set function

The table below describes the function tmr_repetition_counter_set.

Table 606. tmr_repetition_counter_set function

Name	Description
Function name	tmr_repetition_counter_set
Function prototype	void tmr_repetition_counter_set(tmr_type *tmr_x, uint8_t tmr_rpr_value);
Function description	Set repetition period register (rpr)
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR8 or TMR20.
Input parameter 2	tmr_rpr_value: timer repetition period value, ranging from 0x00 to 0xFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_repetition_counter_set(TMR1, 0x10);
```

5.23.10 tmr_counter_value_set function

The table below describes the function tmr_counter_value_set.

Table 607. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	tmr_cnt_value: timer counter value, 0x0000~0xFFFF for 16-bit timer, and 0x0000_0000~0xFFFF_FFFF for 32-timer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_counter_value_set(TMR1, 0xFFFF);
```

5.23.11 tmr_counter_value_get function

The table below describes the function tmr_counter_value_get.

Table 608. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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);
```

5.23.12 tmr_div_value_set function

The table below describes the function tmr_div_value_set.

Table 609. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	tmr_div_value: timer frequency division value, 0x0000~0xFFFF for 16-bit timer and 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);
```

5.23.13 tmr_div_value_get function

The table below describes the function tmr_div_value_get.

Table 610. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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);
```

5.23.14 tmr_output_channel_config function

The table below describes the function tmr_output_channel_config.

Table 611. 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 channel
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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

Set 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_SELECT_CHANNEL_5: Channel 5

tmr_output_config_type structure

The tmr_output_config_type is defined in the at32f435_437_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 mode A
TMR_OUTPUT_CONTROL_PWM_MODE_B: PWM mode B

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);
```

5.23.15 tmr_output_channel_mode_select function

The table below describes the function tmr_output_channel_mode_select.

Table 612. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	oc_mode: output mode
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
TMR_SELECT_CHANNEL_5: Timer channel 5

oc_mode

Set output channel mode, that is, to configure channel original signals (CxORAW)

TMR_OUTPUT_CONTROL_OFF: Disconnect channel output (CxOUT) fromCxORAW
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 mode A
TMR_OUTPUT_CONTROL_PWM_MODE_B: PWM mode B

Example:

```
tmr_output_channel_mode_select(TMR1, TMR_SELECT_CHANNEL_1, TMR_OUTPUT_CONTROL_SWITCH);
```

5.23.16 tmr_period_value_set function

The table below describes the function tmr_period_value_set.

Table 613. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	tmr_pr_value: timer period value, 0x0000~0xFFFF for 16-bit timer and 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);
```

5.23.17 tmr_period_value_get function

The table below describes the function tmr_period_value_get.

Table 614. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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);
```

5.23.18 tmr_channel_value_set function

The table below describes the function tmr_channel_value_set.

Table 615. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	tmr_channel_value: timer channel value, 0x0000~0xFFFF for 16-bit timer and 0x0000_0000~0xFFFF_FFFF for 32-bit timer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Set 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
TMR_SELECT_CHANNEL_5: Timer channel 5

Example:

```
tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_1, 0xFFFF);
```

5.23.19 tmr_channel_value_get function

The table below describes the function tmr_channel_value_get.

Table 616. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	tmr_channel: timer channel
Output parameter	Timer channel value
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Set 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
TMR_SELECT_CHANNEL_5: Timer channel 5

Example:

```
uint32_t ch_value;
ch_value = tmr_channel_value_get(TMR1, TMR_SELECT_CHANNEL_1);
```

5.23.20 tmr_period_buffer_enable function

The table below describes the function tmr_period_buffer_enable.

Table 617. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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);
```

5.23.21 tmr_output_channel_buffer_enable function

The table below describes the function tmr_output_channel_buffer_enable.

Table 618. tmr_output_channel_buffer_enable function

Name	Description
Function name	tmr_output_channel_buffer_enable
Function prototype	void tmr_output_channel_buffer_enable(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, confirm_state new_state);
Function description	Enable or disable TMR output channel buffer
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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).

Name	Description
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
TMR_SELECT_CHANNEL_5: Timer channel 5

Example:

```
tmr_output_channel_buffer_enable(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

5.23.22 tmr_output_channel_immediately_set function

The table below describes the function tmr_output_channel_immediately_set.

Table 619. tmr_output_channel_immediately_set function

Name	Description
Function name	tmr_output_channel_immediately_set
Function prototype	void tmr_output_channel_immediately_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, confirm_state new_state);
Function description	Enable TMR output channel immediately
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	new_state: indicates the status of output channel enable. 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
TMR_SELECT_CHANNEL_5: Timer channel 5

Example:

```
tmr_output_channel_immediately_set(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

5.23.23 tmr_output_channel_switch_set function

The table below describes the function tmr_output_channel_switch_set.

Table 620. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	new_state: indicates the status of output channel switch. 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
TMR_SELECT_CHANNEL_5: Timer channel 5

Example:

```
tmr_output_channel_switch_set(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

5.23.24 tmr_one_cycle_mode_enable function

The table below describes the function tmr_one_cycle_mode_enable.

Table 621. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	new_state: indicates the status of one-cycle mode. It 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);
```

5.23.25 tmr_32_bit_function_enable function

The table below describes the function tmr_32_bit_function_enable.

Table 622. 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 mode (plus mode)
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR2 or TMR5.
Input parameter 2	new_state: the status of 32-bit mode It 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);
```

5.23.26 tmr_overflow_request_source_set function

The table below describes the function tmr_overflow_request_source_set.

Table 623. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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);
```

5.23.27 tmr_overflow_event_disable function

The table below describes the function tmr_overflow_event_disable.

Table 624. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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);
```

5.23.28 tmr_input_channel_init function

The table below describes the function tmr_input_channel_init.

Table 625. 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 channel
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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

The tmr_input_config_type is defined in the at32f435_437_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 channels 1, 2, 3 and 4 are linked to C1IRAW, C2IRAW, C3IRAW and C4IRAW respectively

TMR_CC_CHANNEL_MAPPED_INDIRECT:
TMR input channels 1, 2, 3 and 4 are 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, ranging from 0x00 to 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);
```

5.23.29 tmr_channel_enable function

The table below describes the function tmr_channel_enable.

Table 626. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	new_state: indicates the status of timer channel. It can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Set 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);
```

5.23.30 tmr_input_channel_filter_set function

The table below describes the function tmr_input_channel_filter_set.

Table 627. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	tmr_channel: timer channel
Input parameter 3	filter_value: set channel filter value, 0x00~0x0F

Name	Description
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);
```

5.23.31 tmr_pwm_input_config function

The table below describes the function tmr_pwm_input_config.

Table 628. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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

tmr_input_config_type pointer; refer to *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);
```

5.23.32 tmr_channel1_input_select function

The table below describes the function tmr_channel1_input_select.

Table 629. tmr_channel1_input_select function

Name	Description
Function name	tmr_channel1_input_select
Function prototype	void tmr_channel1_input_select(tmr_type *tmr_x, tmr_channel1_input_connected_type ch1_connect);
Function description	Select TMR channel 1 input
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8 or TMR20.
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_CHANEL1_CONNECTED_C1IRAW: CH1 pin is connected to C1IRAW

TMR_CHANEL1_2_3_CONNECTED_C1IRAW_XOR: Connect XOR results of CH1, CH2 and CH3 pins to C1IRAW

Example:

```
tmr_channel1_input_select(TMR1, TMR_CHANEL1_2_3_CONNECTED_C1IRAW_XOR);
```

5.23.33 tmr_input_channel_divider_set function

The table below describes the function tmr_input_channel_divider_set.

Table 630. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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);
```

5.23.34 tmr_primary_mode_select function

The table below describes the function tmr_primary_mode_select

Table 631. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8 or TMR20.
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);
```

5.23.35 tmr_sub_mode_select function

The table below describes the function tmr_sub_mode_select.

Table 632. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR12 or TMR20.
Input parameter 2	sub_mode: slave timer mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

primary_mode

Select slave timer modes.

TMR_SUB_MODE_DIABLE: Disable
TMR_SUB_ENCODER_MODE_A: Encoder mode A
TMR_SUB_ENCODER_MODE_B: Encoder mode B
TMR_SUB_ENCODER_MODE_C: Encoder mode C
TMR_SUB_RESET_MODE: Reset
TMR_SUB_HANG_MODE: Suspend
TMR_SUB_TRIGGER_MODE: Trigger
TMR_SUB_EXTERNAL_CLOCK_MODE_A: External clock A

Example:

```
tmr_sub_mode_select(TMR1, TMR_SUB_HANG_MODE);
```

5.23.36 tmr_channel_dma_select function

The table below describes the function tmr_channel_dma_select.

Table 633. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR12 or TMR20.
Input parameter 2	cc_dma_select: TMR channel DMA request source
Output parameter	NA
Return value	NA

Name	Description
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);
```

5.23.37 tmr_hall_select function

The table below describes the function tmr_hall_select.

Table 634. 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: selected TMR peripheral. This parameter can be TMR1, TMR8 or TMR20.
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);
```

5.23.38 tmr_channel_buffer_enable function

The table below describes the function tmr_channel_buffer_enable.

Table 635. 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: selected TMR peripheral. This parameter can be TMR1, TMR8 or TMR20.
Input parameter 2	new_state: indicates the status of TMR channel buffer. It can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA

Name	Description
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_channel_buffer_enable(TMR1, TRUE);
```

5.23.39 tmr_trgout2_enable function

The table below describes the function tmr_trgout2_enable.

Table 636. tmr_trgout2_enable function

Name	Description
Function name	tmr_trgout2_enable
Function prototype	void tmr_trgout2_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR trigger output 2 signal
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR8 or TMR20.
Input parameter 2	new_state: indicates the status of TMR trigger output 2 signal. It can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_trgout2_enable(TMR1, TRUE);
```

5.23.40 tmr_trigger_input_select function

The table below describes the function tmr_trigger_input_select.

Table 637. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR12 or TMR20.
Input parameter 2	trigger_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);
```

5.23.41 tmr_sub_sync_mode_set function

The table below describes the function tmr_sub_sync_mode_set.

Table 638. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR12 or TMR20.
Input parameter 2	new_state: indicates the status of TMR slave timer synchronization mode It 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);
```

5.23.42 tmr_dma_request_enable function

The table below describes the function tmr_dma_request_enable.

Table 639. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	dma_request: DMA request
Input parameter 3	new_state: indicates the status of DMA request

Name	Description
	It 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);
```

5.23.43 tmr_interrupt_enable function

The table below describes the function tmr_interrupt_enable.

Table 640. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	tmr_interrupt: TMR interrupts
Input parameter 3	new_state: indicates the status of TMR interrupts It can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_interrupt

Select a TMR interrupt

TMR_OVF_INT: Overflow event interrupt
TMR_C1_INT: Channel 1 event interrupt
TMR_C2_INT: Channel 2 event interrupt
TMR_C3_INT: Channel 3 event interrupt
TMR_C4_INT: Channel 4 event interrupt
TMR_HALL_INT: HALL event interrupt

TMR_TRIGGER_INT: Trigger event interrupt
TMR_BRK_INT: Break event interrupt

Example:

```
tmr_interrupt_enable(TMR1, TMR_OVF_INT, TRUE);
```

5.23.44 tmr_interrupt_flag_get function

The table below describes the function tmr_interrupt_flag_get.

Table 641. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	tmr_flag: flag selection Refer to the “tmr_flag” below for details.
Output parameter	NA
Return value	flag_status: 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_C5_FLAG: Channel 5 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)
```

5.23.45 tmr_flag_get function

The table below describes the function tmr_flag_get.

Table 642. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	tmr_flag: flag selection Refer to the “tmr_flag” description for details.
Output parameter	NA
Return value	flag_status: status of flag 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_C5_FLAG: Channel 5 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)
```

5.23.46 tmr_flag_clear function

The table below describes the function tmr_flag_clear.

Table 643. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
Input parameter 2	tmr_flag: flag selection Refer to the “tmr_flag” description for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_flag_clear(TMR1, TMR_OVF_FLAG);
```

5.23.47 tmr_event_sw_trigger function

The table below describes the function tmr_event_sw_trigger

Table 644. 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 event
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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);
```

5.23.48 tmr_output_enable function

The table below describes the function tmr_output_enable.

Table 645. 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: selected TMR peripheral. This parameter can be TMR1, TMR8 or TMR20.
Input parameter 2	new_state: TMR output status It 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);
```

5.23.49 tmr_internal_clock_set function

The table below describes the function tmr_internal_clock_set.

Table 646. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR12 or TMR20.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_internal_clock_set(TMR1);
```

5.23.50 tmr_output_channel_polarity_set function

The table below describes the function tmr_output_channel_polarity_set.

Table 647. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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);
```

5.23.51 tmr_external_clock_config function

The table below describes the function tmr_external_clock_config.

Table 648. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8 or TMR20.

Name	Description
Input parameter 2	es_divide: external signal frequency division factor
Input parameter 3	es_polarity: external signal polarity
Input parameter 4	es_filter: external signal filter value, 0x00~0x0F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

es_divide

Set TMR external signal frequency division factor

TMR_ES_FREQUENCY_DIV_1: Divided by 1

TMR_ES_FREQUENCY_DIV_2: Divided by 2

TMR_ES_FREQUENCY_DIV_4: Divided by 4

TMR_ES_FREQUENCY_DIV_8: Divided by 8

es_polarity

Select TMR external signal polarity

TMR_ES_POLARITY_NON_INVERTED: 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);
```

5.23.52 tmr_external_clock_mode1_config function

The table below describes the function tmr_external_clock_mode1_config.

Table 649. tmr_external_clock_mode1_config function

Name	Description
Function name	tmr_external_clock_mode1_config
Function prototype	void tmr_external_clock_mode1_config(tmr_type *tmr_x, tmr_external_signal_divider_type es_divide, tmr_external_signal_polarity_type es_polarity, uint16_t es_filter);
Function description	Configure TMR external clock mode 1 (corresponding to external mode A in the reference manual)
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8 or TMR20.
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);
```

5.23.53 tmr_external_clock_mode2_config function

The table below describes the function tmr_external_clock_mode2_config.

Table 650. tmr_external_clock_mode2_config function

Name	Description
Function name	tmr_external_clock_mode2_config
Function prototype	void tmr_external_clock_mode2_config(tmr_type *tmr_x, tmr_external_signal_divider_type es_divide, tmr_external_signal_polarity_type es_polarity, uint16_t es_filter);
Function description	Configure TMR external clock mode 2 (corresponding to external mode B in the reference manual)
Input parameter 1	tmr_x: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8 or TMR20.
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_mode2_config(TMR1, TMR_ES_FREQUENCY_DIV_1, TMR_ES_POLARITY_INVERTED,
0x0F);
```

5.23.54 tmr_encoder_mode_config function

The table below describes the function tmr_encoder_mode_config.

Table 651. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8 or TMR20.
Input parameter 2	encoder_mode: encoder mode

Name	Description
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);
```

5.23.55 tmr_force_output_set function

The table below describes the function tmr_force_output_set.

Table 652. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14 or TMR20.
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
TMR_SELECT_CHANNEL_5: Timer channel 5

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);
```

5.23.56 tmr_dma_control_config function

The table below describes the function tmr_dma_control_config.

Table 653. 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: selected TMR peripheral. This parameter can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8 or TMR20.
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
TMR_ISTS_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);
```

5.23.57 tmr_brkdt_config function

The table below describes the function tmr_brkdt_config.

Table 654. 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: selected TMR peripheral. This parameter can be TMR1, TMR8 or TMR20.
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 at32f435_437_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, ranging from 0x00 to 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);
```

5.23.58 tmr_iremap_config function

The table below describes the function tmr_iremap_config.

Table 655. 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: selected TMR peripheral. This parameter can be TMR2 or TMR5.
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 TMR2 internal trigger 1 remapping and TMR5 channel 4 input remapping.

TMR2_TMR8TRGOUT_TMR5_GPIO:

TMR2 is connected to TMR8_TRGO output, and TMR5 is connected to GPIO

TMR2_PTP_TMR5_ERTCCLK:

TMR2 is connected to Ethernet PTP output, and TMR5 is connected to LICK

TMR2_OTG1FS_TMR5_LEXT:

TMR2 is connected to OTG1_FS_SOF, and TMR5 is connected to LEXT

TMR2_OTG2FS_TMR5_ERTC:

TMR2 is connected to OTG2_FS_SOF, and TMR5 is connected to ERTC wakeup interrupt

Example:

```
tmr_iremap_config(TMR2, TMR2_TMR8TRGOUT_TMR5_GPIO);
```

5.24 Universal synchronous/asynchronous receiver/transmitter (USART)

The USART register structure usart_type is defined in the “at32f435_437_usart.h”.

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

} usart_type;
```

The table below gives a list of USART registers.

Table 656. 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 register

The table below gives a list of USART library functions.

Table 657. 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
usart_break_send	Send break frame
usart_smartcard_guard_time_set	Set smartcard guard time
usart_irda_smartcard_division_set	Set infrared and smartcard division
usart_smartcard_mode_enable	Enable smartcard mode
usart_smartcard_nack_set	Enable smartcard NACK
usart_single_line_halfduplex_select	Enable single-wire half-duplex mode
usart_irda_mode_enable	Enable infrared mode
usart_irda_low_power_enable	Enable infrared low-power mode
usart_hardware_flow_control_set	Enable hardware flow control
usart_flag_get	Get flag
usart_interrupt_flag_get	Get interrupt flag status
usart_flag_clear	Clear flag
usart_rs485_delay_time_config	Set latency for starting or ending consecutive data transmission in RS485
usart_transmit_receive_pin_swap	Swap transmit/receive pins
usart_id_bit_num_set	Set ID bit count
usart_de_polarity_set	DE signal polarity selection
usart_rs485_mode_enable	RS485 mode enable

5.24.1 usart_reset function

The table below describes the function usart_reset.

Table 658. 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: selected peripherals. It can be USART1, USART2 or 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);
```

5.24.2 usart_init function

The table below describes the function usart_init.

Table 659. 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: selected peripherals. It can be USART1, USART2 or USART3…
Input parameter 2	baud_rate: baud rate for serial interfaces
Input parameter 3	data_bit: data bit width for serial interfaces
Input parameter 4	stop_bit: stop bit width for serial interfaces
Output parameter	NA
Return value	NA
Required preconditions	This operation can be allowed only when external low-speed clock is disabled.
Called functions	NA

data_bit

Select data bit size for serial interface communication

USART_DATA_8BITS: 8-bit

USART_DATA_9BITS: 9-bit

stop_bit

Select stop bit size for serial interface communication

USART_STOP_1_BIT: 1-bit

USART_STOP_0_5_BIT: 0.5-bit

USART_STOP_2_BIT: 2-bit

USART_STOP_1_5_BIT: 1.5-bit

Example:

```
/* configure uart param */
usart_init(USART1, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
```

5.24.3 usart_parity_selection_config function

The table below describes the function usart_parity_selection_config.

Table 660. usart_parity_selection_config function

Name	Description
Function name	usart_parity_selection_config
Function prototype	void usart_parity_selection_config(usart_type* usart_x, usart_parity_selection_type parity);
Function description	Parity selection
Input parameter 1	usart_x: selected peripheral. It can be USART1, USART2 or USART3…
Input parameter 2	parity: parity mode for serial interface communication
Output parameter	NA

Name	Description
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);
```

5.24.4 usart_enable function

The table below describes the function usart_enable.

Table 661. 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: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.5 usart_transmitter_enable function

The table below describes the function usart_transmitter_enable.

Table 662. 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: selected peripheral. It can be USART1, USART2 or USART3…
Input parameter 2	new_state: Enable (TRUE) and disable (FALSE)
Output parameter	NA
Return value	NA

Name	Description
Required preconditions	NA
Called functions	NA

Example:

```
/* enable usart1 transmitter */
usart_transmitter_enable(USART1, TRUE);
```

5.24.6 usart_receiver_enable function

The table below describes the function usart_receiver_enable.

Table 663. 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: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.7 usart_clock_config function

The table below describes the function usart_clock_config.

Table 664. 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_pha, usart_lbcp_type clk_lb);
Function description	Configure clock polarity and phase for synchronization feature
Input parameter 1	usart_x: selected peripheral. It can be USART1, USART2 or USART3…
Input parameter 2	clk_pol: clock polarity for synchronization
Input parameter 3	clk_pha: clock phase for synchronization
Input parameter 4	clk_lb: selects whether to output clock on the last bit (upper bit) of data sent through synchronization feature
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

clk_pol

Clock polarity selection
USART_CLOCK_POLARITY_LOW: Low
USART_CLOCK_POLARITY_HIGH: High

clk_pha

Clock phase selection
USART_CLOCK_PHASE_1EDGE: 1st edge
USART_CLOCK_PHASE_2EDGE: 2nd 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);
```

5.24.8 usart_clock_enable function

The table below describes the function usart_clock_enable.

Table 665. 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: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.9 usart_interrupt_enable function

The table below describes the function usart_interrupt_enable.

Table 666. 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: selected peripheral. It can be USART1, USART2 or USART3…
Input parameter 2	usart_int: interrupt type

Name	Description
Input parameter 3	new_state: enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

usart_int

Select a peripheral interrupt

USART_IDLE_INT: Bus idle

USART_RDBF_INT: Receive data buff full

USART_TDC_INT: Transmit data complete

USART_TDBE_INT: Transmit data buff 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);
```

5.24.10 usart_dma_transmitter_enable function

The table below describes the function usart_dma_transmitter_enable.

Table 667. 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: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.11 usart_dma_receiver_enable function

The table below describes the function usart_dma_receiver_enable.

Table 668. 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: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.12 usart_wakeup_id_set function

The table below describes the function usart_wakeup_id_set.

Table 669. 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: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.13 usart_wakeup_mode_set function

The table below describes the function usart_wakeup_mode_set.

Table 670. 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: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.14 usart_receiver_mute_enable function

The table below describes the function usart_receiver_mute_enable.

Table 671. 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: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.15 usart_break_bit_num_set function

The table below describes the function usart_break_bit_num_set.

Table 672. 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: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.16 usart_lin_mode_enable function

The table below describes the function usart_lin_mode_enable.

Table 673. 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: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.17 usart_data_transmit function

The table below describes the function usart_data_transmit.

Table 674. 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: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.18 usart_data_receive function

The table below describes the function usart_data_receive.

Table 675. usart_data_receive function

Name	Description
Function name	usart_data_receive
Function prototype	uint16_t usart_data_receive(usart_type* usart_x);
Function description	Receive data
Input parameter 1	usart_x: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.19 usart_break_send function

The table below describes the function usart_break_send.

Table 676. usart_break_send function

Name	Description
Function name	usart_break_send
Function prototype	void usart_break_send(usart_type* usart_x);
Function description	Send break frame
Input parameter 1	usart_x: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.20 usart_smartcard_guard_time_set function

The table below describes the function usart_smartcard_guard_time_set.

Table 677. 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: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.21 usart_irda_smartcard_division_set function

The table below describes the function usart_irda_smartcard_division_set.

Table 678. 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: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.22 usart_smartcard_mode_enable function

The table below describes the function usart_smartcard_mode_enable.

Table 679. 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: selected peripheral. It can be USART1, USART2 or 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 the smartcard mode */
usart_smartcard_mode_enable(USART1, TRUE);
```

5.24.23 usart_smartcard_nack_set function

The table below describes the function usart_smartcard_nack_set.

Table 680. 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: selected peripheral. It can be USART1, USART2 or 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 the nack transmission */
usart_smartcard_nack_set(USART1, TRUE);
```

5.24.24 usart_single_line_halfduplex_select function

The table below describes the function usart_single_line_halfduplex_select.

Table 681. 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: selected peripheral. It can be USART1, USART2 or 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 halfduplex */
usart_single_line_halfduplex_select(USART1, TRUE);
```

5.24.25 usart_irda_mode_enable function

The table below describes the function usart_irda_mode_enable.

Table 682. 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: selected peripheral. It can be USART1, USART2 or 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 irda mode */
usart_irda_mode_enable(USART1, TRUE);
```

5.24.26 usart_irda_low_power_enable function

The table below describes the function usart_irda_low_power_enable.

Table 683. 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: selected peripheral. It can be USART1, USART2 or 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 irda lowpower mode */
usart_irda_low_power_enable (USART1, TRUE);
```

5.24.27 usart_hardware_flow_control_set function

The table below describes the function usart_hardware_flow_control_set.

Table 684. 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: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.28 usart_flag_get function

The table below describes the function usart_flag_get.

Table 685. 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: selected peripheral. It can be USART1, USART2 or 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 buff empty flag
USART_TDC_FLAG: Transmit complete flag
USART_RDBF_FLAG: Receive data buff 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);
```

5.24.29 usart_interrupt_flag_get function

The table below describes the function usart_interrupt_flag_get.

Table 686. 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: selected peripheral. It can be USART1, USART2…
Input parameter 2	flag: select a 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 buff empty flag
USART_TDC_FLAG: Transmit complete flag
USART_RDBF_FLAG: Receive data buff full flag
USART_IDLEF_FLAG: Idle frame flag
USART_ROERR_FLAG: Receive overflow flag
USART_NERR_FLAG: Noise error flag
USART_FERR_FLAG: Frame error flag
USART_PERR_FLAG: Parity error flag

Example:

```
/* check received data flag */
if(usart_interrupt_flag_get(USART1, USART_RDBF_FLAG) != RESET)
{
}
```

5.24.30 usart_flag_clear function

The table below describes the function usart_flag_clear.

Table 687. 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: selected peripheral. It can be USART1, USART2 or USART3…
Input parameter 2	flag: flag
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flag

USART_CTSCF_FLAG: CTS (Clear To Send) change flag
USART_BFF_FLAG: Break frame receive flag
USART_TDC_FLAG: Transmit complete flag
USART_RDBF_FLAG: Receive data buff full flag

Example:

```
/* clear data transmit complete flag */
usart_flag_clear (USART1, USART_TDC_FLAG );
```

5.24.31 usart_rs485_delay_time_config function

The table below describes the function usart_rs485_delay_time_config.

Table 688. usart_rs485_delay_time_config 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: selected peripheral. It can be USART1, USART2 or USART3…
Input parameter 2	start_delay_time: 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: 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);
```

5.24.32 usart_transmit_receive_pin_swap function

The table below describes the function usart_transmit_receive_pin_swap.

Table 689. 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: selected peripheral. It can be USART1, USART2 or 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 tx/rx swap */
usart_transmit_receive_pin_swap (USART1, TRUE);
```

5.24.33 usart_id_bit_num_set function

The table below describes the function usart_id_bit_num_set.

Table 690. usart_id_bit_num_set 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: selected peripheral. It can be USART1, USART2 or USART3…
Input parameter 2	id_bit_num: ID bit number
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);
```

5.24.34 usart_de_polarity_set function

The table below describes the function usart_de_polarity_set.

Table 691. 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: selected peripheral. It can be USART1, USART2 or 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);
```

5.24.35 usart_rs485_mode_enable function

The table below describes the function usart_rs485_mode_enable.

Table 692. 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: selected peripheral. It can be USART1, USART2 or 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 rs485 mode */
usart_rs485_mode_enable (USART1, TRUE);
```

5.25 Watchdog timer (WDT)

The WDT register structure wdt_type is defined in the “at32f435_437_wdt.h”.

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


} wdt_type;
```

The table below gives a list of WDT registers.

Table 693. 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 694. Summary WDT library functions

Function name	Description
wdt_enable	Enable watchdog
wdt_counter_reload	Reload counter
wdt_reload_value_set	Set reload value
wdt_divider_set	Set division value
wdt_register_write_enable	Unlock WDT_DIV, WDT_RLD and WDT_WIN register write protection
wdt_flag_get	Get flag
wdt_window_counter_set	Set window value

5.25.1 wdt_enable function

The table below describes the function wdt_enable.

Table 695. wdt_enable function

Name	Description
Function name	wdt_enable
Function prototype	void wdt_enable(void);
Function description	Enable watchdog
Input parameter 1	NA
Output parameter	NA
Return value	NA
Required preconditions	NA

Name	Description
Called functions	NA

Example:

```
wdt_enable();
```

5.25.2 wdt_counter_reload function

The table below describes the function wdt_counter_reload.

Table 696. wdt_counter_reload function

Name	Description
Function name	wdt_counter_reload
Function prototype	void wdt_counter_reload(void);
Function description	Reload counter
Input parameter 1	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wdt_counter_reload();
```

5.25.3 wdt_reload_value_set function

The table below describes the function wdt_reload_value_set.

Table 697. wdt_reload_value_set function

Name	Description
Function name	wdt_reload_value_set
Function prototype	void wdt_reload_value_set(uint16_t reload_value);
Function description	Set reload value
Input parameter 1	reload_value: reload value, 0x000~0xFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wdt_reload_value_set(0xFFF);
```

5.25.4 wdt_divider_set function

The table below describes the function wdt_divider_set.

Table 698. wdt_divider_set function

Name	Description
Function name	wdt_divider_set
Function prototype	void wdt_divider_set(wdt_division_type division);
Function description	Set division value
Input parameter 1	division: watchdog division value Refer to the following “division” descriptions 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);
```

5.25.5 wdt_register_write_enable function

The table below describes the function wdt_register_write_enable.

Table 699. 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, WDT_RLD and WDT_WIN write protection
Input parameter 1	new_state: unlock register write protection This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wdt_register_write_enable(TRUE);
```

5.25.6 wdt_flag_get function

The table below describes the function wdt_flag_get.

Table 700. 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 status
Input parameter 1	flag: flag selection Refer to the “flag” description for details.
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

flag

This is used for flag selection, including

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);
```

5.25.7 wdt_window_counter_set function

The table below describes the function wdt_window_counter_set.

Table 701. wdt_window_counter_set function

Name	Description
Function name	wdt_window_counter_set
Function prototype	void wdt_window_counter_set(uint16_t window_cnt);
Function description	Set window value
Input parameter 1	window_cnt: window value, 0x000~0xFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wdt_window_counter_set(0x7FF);
```

5.26 Window watchdog timer (WWDT)

The WWDT register structure wwdt_type is defined in the “at32f435_437_wwdt.h”.

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


} wwdt_type;
```

The table below gives a list of WWDT registers.

Table 702. 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 703. 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

5.26.1 wwdt_reset function

The table below describes the function wwdt_reset.

Table 704. 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 1	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();
```

5.26.2 wwdt_divider_set function

The table below describes the function wwdt_divider_set.

Table 705. 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 1	division: WWDT division value Refer to the following “division” descriptions 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);
```

5.26.3 wwdt_enable function

The table below describes the function wwdt_enable.

Table 706. wwdt_enable function

Name	Description
Function name	wwdt_enable
Function prototype	void wwdt_enable(uint8_t wwdt_cnt);
Function description	Enable WWDT
Input parameter 1	wwdt_cnt: WWDT counter initial value, 0x40~0x7F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_enable(0x7F);
```

5.26.4 wwdt_interrupt_enable function

The table below describes the function wwdt_interrupt_enable.

Table 707. 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 1	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_interrupt_enable();
```

5.26.5 wwdt_counter_set function

The table below describes the function wwdt_counter_set.

Table 708. 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 1	wwdt_cnt: WWDT counter value, 0x40~0x7F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_counter_set(0x7F);
```

5.26.6 wwdt_window_counter_set function

The table below describes the function wwdt_window_counter_set.

Table 709. wwdt_window_counter_set function

Name	Description
Function name	wwdt_window_counter_set
Function prototype	void wwdt_window_counter_set(uint8_t window_cnt);
Function description	Set window value
Input parameter 1	wwdt_cnt: WWDT window value, 0x40~0x7F
Output parameter	NA
Return value	NA
Required preconditions	NA

Name	Description
Called functions	NA

Example:

```
wwdt_window_counter_set(0x6F);
```

5.26.7 wwdt_flag_get function

The table below describes the function wwdt_flag_get.

Table 710. 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 1	NA
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_flag_get();
```

5.26.8 wwdt_interrupt_flag_get function

The table below describes the function wwdt_ interrupt_flag_get.

Table 711. 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 1	NA
Output parameter	NA
Return value	flag_status: SET or RESET
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_ interrupt_flag_get();
```

5.26.9 wwdt_flag_clear function

The table below describes the function wwdt_flag_clear.

Table 712. 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 1	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_flag_clear();
```

5.27 External memory controller (XMC)

For XMC bank1 chip-select control register and timing register, their register structure xmc_bank1_ctrl_tmg_reg_type is defined in the “at32f435_437_xmc.h”.

```
/**
  * @brief type define xmc bank1 ctrl and tmg register
  */
typedef struct
{
      …

} xmc_bank1_ctrl_tmg_reg_type;
```

XMC bank1 write timing register structure xmc_bank1_tmgwr_reg_type is defined in the **“**at32f435_437_xmc.h**”.**

```
/**
  * @brief type define xmc bank1 tmgwr register
  */
typedef struct
{
      …

} xmc_bank1_tmgwr_reg_type;
```

XMC bank1 register structure xmc_bank1_type is defined in the “at32f435_437_xmc.h”.

```
/**
  * @brief xmc bank1 registers
  */
typedef struct
{
      …

} xmc_bank1_type;
```

XMC bank2 register structure xmc_bank2_type is defined in the “at32f435_437_xmc.h”.

```
/**
  * @brief xmc bank2 registers
  */
typedef struct
{
      …

} xmc_bank2_type;
```

XMC bank3 register structure xmc_bank3_type is defined in the “at32f435_437_xmc.h”.

```
/**
  * @brief xmc bank3 registers
```

```
  */
typedef struct
{
      …

} xmc_bank3_type;
```

XMC bank4 register structure xmc_bank4_type is defined in the “at32f435_437_xmc.h”.

```
/**
  * @brief xmc bank4 registers
  */
typedef struct
{
      …

} xmc_bank4_type;
```

XMC sdram register structure xmc_sdram_type is defined in the “at32f435_437_xmc.h”.

```
/**
  * @brief xmc sdram type
  */
typedef struct
{
      …
} xmc_sdram_type
```

The table below gives a list of XMC registers.

Table 713. Summary of XMC registers

Register	Description
xmc_bk1ctrl1	SRAM/NOR Flash chip select control register 1
xmc_bk1tmg1	SRAM/NOR Flash chip select timing register 1
xmc_bk1ctrl2	SRAM/NOR Flash chip select control register 2
xmc_bk1tmg2	SRAM/NOR Flash chip select timing register 2
xmc_bk1ctrl3	SRAM/NOR Flash chip select control register 3
xmc_bk1tmg3	SRAM/NOR Flash chip select timing register 3
xmc_bk1ctrl4	SRAM/NOR Flash chip select control register 4
xmc_bk1tmg4	SRAM/NOR Flash chip select timing register 4
xmc_bk2ctrl	Nand Flash control register 2
xmc_bk2is	Interrupt enable and FIFO status register 2
xmc_bk2tmgrg	Regular memory timing register 2
xmc_bk2tmgsp	Special memory timing register 2
xmc_bk2ecc	ECC value register 2
xmc_bk3ctrl	Nand Flash control register 3
xmc_bk3is	Interrupt enable and FIFO status register 3
xmc_bk3tmgrg	Regular memory timing register 3

Register	Description
xmc_bk3tmgsp	Special memory timing register 3
xmc_bk3ecc	ECC value register 3
xmc_bk4ctrl	PC card control register
xmc_bk4is	Interrupt enable and FIFO status register 4
xmc_bk4tmgcm	Common memory timing register 4
xmc_bk4tmgat	Attribute memory timing register 4
xmc_bk4tmgio	IO space timing register 4
xmc_bk1tmgwr1	SRAM/NOR Flash write timing register 1
xmc_bk1tmgwr2	SRAM/NOR Flash write timing register 2
xmc_bk1tmgwr3	SRAM/NOR Flash write timing register 3
xmc_bk1tmgwr4	SRAM/NOR Flash write timing register 4
xmc_ext1	SRAM/NOR Flash extra timing register 1
xmc_ext2	SRAM/NOR Flash extra timing register 2
xmc_ext3	SRAM/NOR Flash extra timing register 3
xmc_ext4	SRAM/NOR Flash extra timing register 4
sdram_ctrl1	SDRAM control register 1
sdram_ ctrl2	SDRAM control register 2
sdram_tm1	SDRAM timing register 1
sdram_tm2	SDRAM timing register 2
sdram_cmd	SDRAM command register
sdram_rcnt	SDRAM refresh timer register
sdram_sts	SDRAM status register

The table below gives a list of XMC library functions.

Table 714. Summary of XMC library functions

Function name	Description
xmc_nor_sram_reset	Reset nor/sram controller
xmc_nor_sram_init	Initialize nor/sram controller
xmc_nor_sram_timing_config	Configure nor/sram controller timing
xmc_norsram_default_para_init	Initialize paramters in xmc_nor_sram_init_struct
xmc_norsram_timing_default_para_init	Initialize paramters in xmc_rw_timing_struct and xmc_w_timing_struct
xmc_nor_sram_enable	Enable nor/sram controller
xmc_ext_timing_config	Configure nor/sram extension controller
xmc_nand_reset	Reset nand controller
xmc_nand_init	Initialize nand controller
xmc_nand_timing_config	Configure nand controller timing
xmc_nand_default_para_init	Initialize paramters in xmc_nand_init_struct
xmc_nand_timing_default_para_init	Initialize paramters in xmc_common_spacetiming_struct and xmc_attribute_spacetiming_struct
xmc_nand_enable	Enable nand controller
xmc_nand_ecc_enable	Enable nand controller ECC feature
xmc_ecc_get	Get ECC value
xmc_interrupt_enable	Enable XMC controller interrupt

xmc_flag_status_get	Get XMC controller flag
xmc_interrupt_flag_status_get	Get XMC controller interrupt flag status
xmc_flag_clear	Clear XMC controller interrupt
xmc_pccard_reset	Reset pccard controller
xmc_pccard_init	Initialize pccard controller
xmc_pccard_timing_config	Configure pccard controller timing
xmc_pccard_default_para_init	Initialize parameters in xmc_pccard_init_struct
xmc_pccard_timing_default_para_init	Initialize parameters in xmc_common_spacetiming_struct, xmc_attribute_spacetiming_struct and xmc_iospace_timing_struc
xmc_pccard_enable	Enable pccard controller
xmc_sdram_reset	Reset sdram bank controller
xmc_sdram_init	Initialize sdram bank controller
xmc_sdram_default_para_init	Initialize parameters in xmc_sdram_init_struct and xmc_sdram_timing_struct
xmc_sdram_cmd	Send sdram command
xmc_sdram_status_get	Get sdram bank status
xmc_sdram_refresh_counter_set	Configure sdram auto refresh counter
xmc_sdram_auto_refresh_set	Set auto refresh times

5.27.1 xmc_nor_sram_reset function

The table below describes the function xmc_nor_sram_reset.

Table 715. xmc_nor_sram_reset function

Name	Description
Function name	xmc_nor_sram_reset
Function prototype	void xmc_nor_sram_reset(xmc_nor_sram_subbank_type xmc_subbank);
Function description	Reset nor/sram controller
Input parameter 1	xmc_subbank: selected subbank
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

xmc_subbank

Select a subbank.

XMC_BANK1_NOR_SRAM1: xmc subbank1

XMC_BANK1_NOR_SRAM2: xmc subbank2

XMC_BANK1_NOR_SRAM3: xmc subbank3

XMC_BANK1_NOR_SRAM4: xmc subbank4

Example:

```
/* reset nor/sram subbank1 */
xmc_nor_sram_reset(XMC_BANK1_NOR_SRAM1);
```

5.27.2 xmc_nor_sram_init function

The table below describes the function xmc_nor_sram_init.

Table 716. xmc_nor_sram_init function

Name	Description
Function name	xmc_nor_sram_init
Function prototype	void xmc_nor_sram_init(xmc_norsram_init_type* xmc_norsram_init_struct);
Function description	Initialize nor/sram controller
Input parameter 1	xmc_norsram_init_struct: xmc_norsram_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

xmc_norsram_init_type struct

The xmc_norsram_init_type is defined in the at32f435_437_xmc.h.

```
typedef struct
{
  xmc_nor_sram_subbank_type        subbank;
  xmc_data_addr_mux_type           data_addr_multiplex;
  xmc_memory_type                  device;
  xmc_data_width_type              bus_type;
  xmc_burst_access_mode_type       burst_mode_enable;
  xmc_asyn_wait_type               asynwait_enable;
  xmc_wait_signal_polarity_type    wait_signal_lv;
  xmc_wrap_mode_type               wrapped_mode_enable;
  xmc_wait_timing_type             wait_signal_config;
  xmc_write_operation_type         write_enable;
  xmc_wait_signal_type             wait_signal_enable;
  xmc_extended_mode_type           write_timing_enable;
  xmc_write_burst_type             write_burst_syn;
} xmc_norsram_init_type;
```

subbank

Select a subbank

XMC_BANK1_NOR_SRAM1: xmc subbank1
XMC_BANK1_NOR_SRAM2: xmc subbank2
XMC_BANK1_NOR_SRAM3: xmc subbank3
XMC_BANK1_NOR_SRAM4: xmc subbank4

data_addr_multiplex

Define whether the lower 16 bits of the xmc address line is multiplexed with data line or not.

XMC_DATA_ADDR_MUX_DISABLE: Not multiplexed
XMC_DATA_ADDR_MUX_ENABLE: Multiplexed

Device

Select an external memory device

XMC_DEVICE_SRAM: sram

XMC_DEVICE_PSRAM: psram
XMC_DEVICE_NOR: nor

bus_type

xmc data bus bit width
XMC_BUSTYPE_8_BITS: 8-bit data bus
XMC_BUSTYPE_16_BITS: 16-bit data bus

burst_mode_enable

Enable or disable burst mode
XMC_BURST_MODE_DISABLE: Burst mode disabled
XMC_BURST_MODE_ENABLE: Burst mode enabled

asynwait_enable

Enable or disable wait signals during asynchronous transmission
XMC_ASYN_WAIT_DISABLE: Disabled
XMC_ASYN_WAIT_ENABLE: Enabled

wait_signal_lv

Select the polarity of wait signals. It is used to set the polarity of NWAIT signal in synchronous mode
XMC_WAIT_SIGNAL_LEVEL_LOW: Active low
XMC_WAIT_SIGNAL_LEVEL_HIGH: Active high

wrapped_mode_enable

In synchronous mode, this bit is used to define whether the XMC controller will or not split a not-aligned AHB access into two accesses.
XMC_WRAPPED_MODE_DISABLE: Direct wrapped burst mode is disabled
XMC_WRAPPED_MODE_ENABLE: Direct wrapped burst mode is enabled

wait_signal_config

Wait timing configuration, valid only in synchronous mode
XMC_WAIT_SIGNAL_SYN_BEFORE: NWAIT signal is active one data cycle before wait state
XMC_WAIT_SIGNAL_SYN_DURING : NWAIT signal is active during wait state

write_enable

Write enable bit
XMC_WRITE_OPERATION_DISABLE: Write operation is disabled
XMC_WRITE_OPERATION_ENABLE: Write operation is enabled

wait_signal_enable

Wait signal enable bit during synchronous transmission
XMC_WAIT_SIGNAL_DISABLE: NWAIT signal disabled
XMC_WAIT_SIGNAL_ENABLE: NWAIT signal enabled

write_timing_enable

Wait timing enable bit. Read memory and write memory operate at different timings. In other words, SRAM/NOR write timing register (XMC_BKxTMGWR) is released.
XMC_WRITE_TIMING_DISABLE: Read and write timings are the same
XMC_WRITE_TIMING_ENABLE: Read and write timings are different

write_burst_syn

Write burst enable bit
XMC_WRITE_BURST_SYN_DISABLE: Write operation is performed in asynchronous mode
XMC_WRITE_BURST_SYN_ENABLE: Write operation is performed in synchronous mode

Example:

```
xmc_norsram_init_type xmc_norsram_init_struct;
```

```
xmc_norsram_init_struct.subbank                 = XMC_BANK1_NOR_SRAM1;
xmc_norsram_init_struct.data_addr_multiplex     = XMC_DATA_ADDR_MUX_ENABLE;
xmc_norsram_init_struct.device                  = XMC_DEVICE_PSRAM;
xmc_norsram_init_struct.bus_type                = XMC_BUSTYPE_16_BITS;
xmc_norsram_init_struct.burst_mode_enable       = XMC_BURST_MODE_DISABLE;
xmc_norsram_init_struct.asynwait_enable         = XMC_ASYN_WAIT_DISABLE;
xmc_norsram_init_struct.wait_signal_lv          = XMC_WAIT_SIGNAL_LEVEL_LOW;
xmc_norsram_init_struct.wrapped_mode_enable     = XMC_WRAPPED_MODE_DISABLE;
xmc_norsram_init_struct.wait_signal_config      = XMC_WAIT_SIGNAL_SYN_BEFORE;
xmc_norsram_init_struct.write_enable            = XMC_WRITE_OPERATION_ENABLE;
xmc_norsram_init_struct.wait_signal_enable      = XMC_WAIT_SIGNAL_DISABLE;
xmc_norsram_init_struct.write_burst_syn         = XMC_WRITE_BURST_SYN_DISABLE;
xmc_norsram_init_struct.write_timing_enable     = XMC_WRITE_TIMING_DISABLE;
xmc_nor_sram_init(&xmc_norsram_init_struct);
```

5.27.3 xmc_nor_sram_timing_config function

The table below describes the function xmc_nor_sram_reset.

Table 717.xmc_nor_sram_timing_config function

Name	Description
Function name	xmc_nor_sram_timing_config
Function prototype	void xmc_nor_sram_timing_config(xmc_norsram_timing_init_type* xmc_rw_timing_struct, xmc_norsram_timing_init_type* xmc_w_timing_struct);
Function description	Configure nor/sram controller timing
Input parameter 1	xmc_rw_timing_struct: xmc_norsram_timing_init_type pointer. This structure is used to configure read and write timings when a separate write timing is not enabled.
Input parameter 2	xmc_w_timing_struct: xmc_norsram_timing_init_type pointer. This structure is used to configure write timings when a separate write timing is enabled.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

xmc_norsram_timing_init_type struct

The xmc_norsram_timing_init_type is defined in the at32f435_437_xmc.h.

```
typedef struct
{
  xmc_nor_sram_subbank_type     subbank;
  xmc_extended_mode_type        write_timing_enable;
  uint32_t                      addr_setup_time;
  uint32_t                      addr_hold_time;
  uint32_t                      data_setup_time;
  uint32_t                      bus_latency_time;
  uint32_t                      clk_psc;
```

```
  uint32_t                    data_latency_time;
  xmc_access_mode_type        mode;
} xmc_norsram_timing_init_type;
```

subbank

Select a subbank to reset

XMC_BANK1_NOR_SRAM1: xmc subbank1
XMC_BANK1_NOR_SRAM2: xmc subbank2
XMC_BANK1_NOR_SRAM3: xmc subbank3
XMC_BANK1_NOR_SRAM4: xmc subbank4

write_timing_enable

Wait timing enable bit. Read memory and write memory operate at different timings. In other words, SRAM/NOR write timing register (XMC_BKxTMGWR) is released.

XMC_WRITE_TIMING_DISABLE: Read and write timings are the same
XMC_WRITE_TIMING_ENABLE: Read and write timings are different

addr_setup_time

Address setup time

0000: 0 HCLK clock cycle added
0001: 1 HCLK clock cycle added
……
1111: 15 HCLK clock cycles added

addr_hold_time

Address hold time

0000: 0 HCLK clock cycle added
0001: 1 HCLK clock cycle added
……
1111: 15 HCLK clock cycles added

data_setup_time

Data setup time

0000: 0 HCLK clock cycle added
0001: 1 HCLK clock cycle added
……
1111: 15 HCLK clock cycles added

bus_latency_time

Bus latency time. In order to avoid conflicts on data bus, in multiplexed mode or synchronous mode, the XMC controller inserts latency time into data bus after a single read operation.

0000: 1 HCLK clock cycle added
0001: 2 HCLK clock cycle added
……
1111: 16 HCLK clock cycles added

clk_psc

Clock frequency division factor. Valid only in synchronous mode. It is used to define the clock frequency of the XMC_CLK.

0000: Reserved
0001: XMC_CLK cycles= 2xHCLK clock cycles
0010: XMC_CLK cycles= 3xHCLK clock cycles

……

1111: XMC_CLK cycles= 16xHCLK clock cycles

data_latency_time

Data latency time. Valid only in synchronous mode.

0000: 0 XMC_CLK clock cycle added

0001: 1 XMC_CLK clock cycle added

……

1111: 15 XMC_CLK clock cycles added

Mode

Asynchronous access mode selection bit. Valid only when the RWTD bit is enabled.

00: Mode A

01: Mode B

10: Mode C

11: Mode D

Example:

```
xmc_norsram_timing_init_type    rw_timing_struct, w_timing_struct;

rw_timing_struct.subbank                  = XMC_BANK1_NOR_SRAM1;
rw_timing_struct.mode                     = XMC_ACCESS_MODE_A;
rw_timing_struct.write_timing_enable      = XMC_WRITE_TIMING_DISABLE;
rw_timing_struct.addr_hold_time           = 0x08;
rw_timing_struct.addr_setup_time          = 0x09;
rw_timing_struct.data_setup_time          = 0x0F;
rw_timing_struct.data_latency_time        = 0x0;
rw_timing_struct.bus_latency_time         = 0x0;
rw_timing_struct.clk_psc                  = 0x0;

w_timing_struct.subbank                   = XMC_BANK1_NOR_SRAM1;
w_timing_struct.mode                      = XMC_ACCESS_MODE_A;
w_timing_struct.write_timing_enable       = XMC_WRITE_TIMING_DISABLE;
w_timing_struct.addr_hold_time            = 0x08;
w_timing_struct.addr_setup_time           = 0x09;
w_timing_struct.data_setup_time           = 0x0F;
w_timing_struct.data_latency_time         = 0x0;
w_timing_struct.bus_latency_time          = 0x0;
w_timing_struct.clk_psc                   = 0x0;

xmc_nor_sram_timing_config(&rw_timing_struct, &w_timing_struct);
```

5.27.4 xmc_norsram_default_para_init function

The table below describes the function xmc_norsram_default_para_init.

Table 718. xmc_norsram_default_para_init function

Name	Description
Function name	xmc_norsram_default_para_init
Function prototype	void xmc_norsram_default_para_init(xmc_norsram_init_type* xmc_nor_sram_init_struct);
Function description	Initialize paramters in xmc_nor_sram_init_struct
Input parameter 1	xmc_nor_sram_init_struct: xmc_norsram_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of members in xmc_nor_sram_init_struct.

Table 719.xmc_nor_sram_init_struct default values

Member	Default value
subbank	XMC_BANK1_NOR_SRAM1
data_addr_multiplex	XMC_DATA_ADDR_MUX_ENABLE
device	XMC_DEVICE_SRAM
bus_type	XMC_BUSTYPE_8_BITS
burst_mode_enable	XMC_BURST_MODE_DISABLE
asynwait_enable	XMC_ASYN_WAIT_DISABLE
wait_signal_lv	XMC_WAIT_SIGNAL_LEVEL_LOW
wrapped_mode_enable	XMC_WRAPPED_MODE_DISABLE
wait_signal_config	XMC_WAIT_SIGNAL_SYN_BEFORE
write_enable	XMC_WRITE_OPERATION_ENABLE
wait_signal_enable	XMC_WAIT_SIGNAL_ENABLE
write_timing_enable	XMC_WRITE_TIMING_DISABLE
write_burst_syn	XMC_WRITE_BURST_SYN_DISABLE

Example:

```
xmc_norsram_init_type xmc_norsram_init_struct;
/* fill each xmc_nor_sram_init_struct member with its default value */
xmc_norsram_default_para_init(&xmc_norsram_init_struct);
```

5.27.5 xmc_norsram_timing_default_para_init function

The table below describes the function xmc_norsram_timing_default_para_init.

Table 720.xmc_norsram_timing_default_para_init function

Name	Description
Function name	xmc_norsram_timing_default_para_init
Function prototype	void xmc_norsram_timing_default_para_init(xmc_norsram_timing_init_type* xmc_rw_timing_struct, xmc_norsram_timing_init_type* xmc_w_timing_struct);

Name	Description
Function description	Initialize parameters in xmc_rw_timing_struct and xmc_w_timing_struct
Input parameter 1	xmc_rw_timing_struct: xmc_norsram_timing_init_type pointer xmc_w_timing_struct: xmc_norsram_timing_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of members in xmc_rw_timing_struct and xmc_w_timing_struct.

Table 721. xmc_rw_timing_struct and xmc_w_timing_struct default values

Member	Default value
subbank	XMC_BANK1_NOR_SRAM1
write_timing_enable	XMC_WRITE_TIMING_DISABLE
addr_setup_time	0xF
addr_hold_time	0xF
data_setup_time	0xFF
bus_latency_time	0xF
clk_psc	0xF
data_latency_time	0xF
mode	XMC_ACCESS_MODE_A

Example:

```
xmc_norsram_timing_init_type  rw_timing_struct, w_timing_struct;
/* fill each xmc_rw_timing_struct and xmc_w_timing_struct member with its default value */
xmc_norsram_timing_default_para_init (&xmc_rw_timing_struct，&xmc_w_timing_struct);
```

5.27.6 xmc_nor_sram_enable function

The table below describes the function xmc_nor_sram_enable.

Table 722. xmc_nor_sram_enable function

Name	Description
Function name	xmc_nor_sram_enable
Function prototype	void xmc_nor_sram_enable(xmc_nor_sram_subbank_type xmc_subbank, confirm_state new_state);
Function description	Enable nor/sram controller
Input parameter 1	xmc_subbank: selected xmc subbank
Input parameter 2	new_state: enable or disable controller
Output parameter	NA

Name	Description
Return value	NA
Required preconditions	NA
Called functions	NA

xmc_subbank

Select a subbank to enable

XMC_BANK1_NOR_SRAM1: xmc subbank1

XMC_BANK1_NOR_SRAM2: xmc subbank2

XMC_BANK1_NOR_SRAM3: xmc subbank3

XMC_BANK1_NOR_SRAM4: xmc subbank4

new_state

FALSE: Disable controller

TRUE: Enable controller

Example:

```
/* enable xmc bank1_sram bank */
xmc_nor_sram_enable(XMC_BANK1_NOR_SRAM1, TRUE);
```

5.27.7 xmc_ext_timing_config function

The table below describes the function xmc_ext_timing_config.

Table 723. xmc_ext_timing_config function

Name	Description
Function name	xmc_ext_timing_config
Function prototype	void xmc_ext_timing_config(xmc_nor_sram_subbank_type xmc_sub_bank, uint16_t w2w_timing, uint16_t r2r_timing);
Function description	Configure nor/sram extension controller
Input parameter 1	xmc_subbank: selected xmc subbank
Input parameter 2	w2w_timing: bus turnaround phase duration between two consecutive write operations.
Input parameter 3	r2r_timing: bus turnaround phase duration between two consecutive read operations.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

xmc_subbank

Select a subbank to reset

XMC_BANK1_NOR_SRAM1: xmc subbank1

XMC_BANK1_NOR_SRAM2: xmc subbank2

XMC_BANK1_NOR_SRAM3: xmc subbank3

XMC_BANK1_NOR_SRAM4: xmc subbank4

w2w_timing

This is used to define the bus turnaround phase duration between two consecutive write operations.

00000000: 1 HCLK clock cycle inserted between two consecutive write operations

00000001: 2 HCLK clock cycles inserted between two consecutive write operations

……

00001000: 9 HCLK clock cycles (default value) inserted between two consecutive write operations
……
11111111: 256 HCLK clock cycles inserted between two consecutive write operations

r2r_timing

This is used to define the bus turnaround phase duration between two consecutive read operations.
00000000: 1 HCLK clock cycle inserted between two consecutive read operations
00000001: 2 HCLK clock cycles inserted between two consecutive read operations
……
00001000: 9 HCLK clock cycles (default value) inserted between two consecutive read operations
……
11111111: 256 HCLK clock cycles inserted between two consecutive read operations

Example:

```
/* bus turnaround phase for consecutive read duration and consecutive write duration */
xmc_ext_timing_config(XMC_BANK1_NOR_SRAM1, 0x08, 0x08);
```

5.27.8 xmc_nand_reset function

The table below describes the function xmc_nand_reset.

Table 724. xmc_nand_reset function

Name	Description
Function name	xmc_nand_reset
Function prototype	void xmc_nand_reset(xmc_class_bank_type xmc_bank);
Function description	Reset nand controller
Input parameter 1	xmc_bank: selected nand controller; it can be XMC_BANK2_NAND
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* reset nand flash */
xmc_nand_reset(XMC_BANK2_NAND);
```

5.27.9 xmc_nand_init function

The table below describes the function xmc_nand_init.

Table 725. xmc_nand_init function

Name	Description
Function name	xmc_nand_init
Function prototype	void xmc_nand_init(xmc_nand_init_type* xmc_nand_init_struct);
Function description	Initialize nand controller
Input parameter 1	xmc_nand_init_struct: xmc_nand_init_type pointer
Output parameter	NA
Return value	NA

Name	Description
Required preconditions	NA
Called functions	NA

xmc_nand_init_type struct

The xmc_nand_init_type is defined in the at32f435_437_xmc.h.

```
typedef struct
{
  xmc_class_bank_type      nand_bank;
  xmc_nand_wait_type       wait_enable;
  xmc_data_width_type      bus_type;
  xmc_ecc_enable_type      ecc_enable;
  xmc_ecc_pagesize_type    ecc_pagesize;
  uint32_t                 delay_time_cycle;
  uint32_t                 delay_time_ar;
} xmc_nand_init_type;
```

Nand_bank

Select a nand_bank controller

XMC_BANK2_NAND: bank2 NAND controller

XMC_BANK3_NAND: bank3 NAND controller

wait_enable

Wait enable bit. It is used to enable NAND Flah memory block wait feature.

XMC_WAIT_OPERATION_DISABLE: Disabled

XMC_WAIT_OPERATION_ENABLE: Enabled

bus_type

External memory data width. It is used to define the external NAND Flash data bus width.

XMC_BUSTYPE_8_BITS: 8-bit data bus

XMC_BUSTYPE_16_BITS: 16-bit data bus

ecc_enable

Define xmc data bus bit width

XMC_BUSTYPE_8_BITS: 8-bit data bus

XMC_BUSTYPE_16_BITS: 16-bit data bus

ecc_pagesize

ECC pase size

000: 256 bytes

001: 512 bytes

010: 1024 bytes

011: 2048 bytes

100: 4096 bytes

101: 8192 bytes

delay_time_cycle

Delay time of CLE to RE (from CLE falling edge to RE falling edge)

0000: 1 HCLK clock cycle

……

1111: 16 HCLK clock cycles

delay_time_ar

Delay time of ALE to RE (from ALE falling edge to RE falling edge)

0000: 1 HCLK clock cycle

……

1111: 16 HCLK clock cycles

Example:

```
xmc_nand_init_type nand_init_struct;
nand_init_struct.nand_bank = XMC_BANK2_NAND;
nand_init_struct.wait_enable = XMC_WAIT_OPERATION_DISABLE;
nand_init_struct.bus_type = XMC_BUSTYPE_8_BITS;
nand_init_struct.ecc_enable = XMC_ECC_OPERATION_DISABLE;
nand_init_struct.ecc_pagesize = XMC_ECC_PAGESIZE_2048_BYTES;
nand_init_struct.delay_time_cycle = 0x10;
nand_init_struct.delay_time_ar = 0x10;
/* xmc nand flash configuration */
xmc_nand_init(&nand_init_struct);
```

5.27.10 xmc_nand_timing_config function

The table below describes the function xmc_nand_timing_config.

Table 726. xmc_nand_timing_config function

Name	Description
Function name	xmc_nand_timing_config
Function prototype	void xmc_nand_timing_config(xmc_nand_timinginit_type* xmc_common_spacetiming_struct, xmc_nand_timinginit_type* xmc_attribute_spacetiming_struct);
Function description	Configure nand controller timing
Input parameter 1	xmc_common_spacetiming_struct: xmc_nand_timinginit_type pointer. It indicates regulary memory timing parameters.
Input parameter 2	xmc_ attribute _spacetiming_struct: xmc_nand_timinginit_type pointer. It indicates special memory timing parametes.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

xmc_nand_timinginit_type struct

The xmc_nand_timinginit_type is defined in the at32f435_437_xmc.h.

```
typedef struct
{
  xmc_class_bank_type      class_bank;
  uint32_t                 mem_setup_time;
  uint32_t                 mem_waite_time;
  uint32_t                 mem_hold_time;
```

```
  uint32_t                      mem_hiz_time;
} xmc_nand_timinginit_type;
```

class_bank

Select a nand flash bank

XMC_BANK2_NAND

XMC_BANK3_NAND

mem_setup_time

Regular memory setup time. It defines the address setup time when access to NAND Flash in a regular memory.

00000000: 0 HCLK clock cycle is inserted

00000001: 1 HCLK clock cycle is inserted

……

11111111: 255 HCLK clock cycles are inserted

mem_waite_time

Regular memory wait time. It specifies the regular memory wait time when the XMC_NWE and XMC_NOE are low.

00000000: 0 HCLK clock cycle is inserted

00000001: 1 HCLK clock cycle is inserted

……

11111111: 255 HCLK clock cycles are inserted

mem_hold_time

Regular memory hold time. defines the databus hold time when access to NAND Flash in a regular memory.

00000000: Reserved

00000001: 1 HCLK clock cycle is inserted

……

11111111: 255 HCLK clock cycles are inserted

mem_hiz_time

Regular memory databus High resistance time. defines the databus high resistance duration when write access to NAND Flash is started in a regular space.

00000000: 0 HCLK clock cycle is inserted

00000001: 1 HCLK clock cycle is inserted

……

11111111: 255 HCLK clock cycles are inserted

Example:

```
xmc_regular_spacetimingstruct.class_bank = XMC_BANK2_NAND;
xmc_regular_spacetimingstruct.mem_setup_time = 254;
xmc_regular_spacetimingstruct.mem_hiz_time = 254;
xmc_regular_spacetimingstruct.mem_hold_time = 254;
xmc_regular_spacetimingstruct.mem_waite_time = 254;
xmc_nand_timing_config(&xmc_regular_spacetimingstruct, &xmc_regular_spacetimingstruct);
```

5.27.11 xmc_nand_default_para_init function

The table below describes the function xmc_nand_default_para_init.

Table 727. xmc_nand_default_para_init function

Name	Description
Function name	xmc_nand_default_para_init
Function prototype	void xmc_nand_default_para_init(xmc_nand_init_type* xmc_nand_init_struct);
Function description	Initialize parameters in xmc_nand_init_struct
Input parameter 1	xmc_nand_init_struct: xmc_nand_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of members xmc_nand_init_struct.

Table 728. xmc_nand_init_struct default values

Member	Default value
nand_bank	XMC_BANK2_NAND
wait_enable	XMC_WAIT_OPERATION_DISABLE
bus_type	XMC_BUSTYPE_8_BITS
ecc_enable	XMC_ECC_OPERATION_DISABLE
ecc_pagesize	XMC_ECC_PAGESIZE_256_BYTES
delay_time_cycle	0x0
delay_time_ar	0x0

Example:

```
/* fill each nand_init_struct member with its default value */
xmc_nand_default_para_init(&nand_init_struct);
```

5.27.12 xmc_nand_timing_default_para_init function

The table below describes the function xmc_nand_timing_default_para_init.

Table 729. xmc_nand_timing_default_para_init function

Name	Description
Function name	xmc_nand_timing_default_para_init
Function prototype	void xmc_nand_timing_default_para_init(xmc_nand_timinginit_type* xmc_common_spacetiming_struct, xmc_nand_timinginit_type* xmc_attribute_spacetiming_struct);
Function description	Initialize parameters in xmc_common_spacetiming_struct and xmc_attribute_spacetiming_struct
Input parameter 1	xmc_common_spacetiming_struct: xmc_nand_timinginit_type pointer xmc_attribute_spacetiming_struct: xmc_nand_timinginit_type pointer
Output parameter	NA
Return value	NA
Required	NA

Name	Description
preconditions	
Called functions	NA

The table below describes the default values of members xmc_rw_timing_struct, xmc_common_spacetiming_struct and xmc_w_timing_struct, xmc_attribute_spacetiming_struct.

Table 730. xmc_common_spacetiming_struct and xmc_attribute_spacetiming_struct default values

Member	Default values
class_bank	XMC_BANK2_NAND
mem_hold_time	0xFC
mem_waite_time	0xFC
mem_setup_time	0xFC
mem_hiz_time	0xFC

Example:

```
xmc_nand_timinginit_type  xmc_regular_spacetimingstruct;
/* fill each xmc_regular_spacetimingstruct member with its default value */
xmc_nand_timing_default_para_init(&xmc_regular_spacetimingstruct, &xmc_regular_spacetimingstruct);
```

5.27.13 xmc_nand_enable function

The table below describes the function xmc_nand_enable.

Table 731. xmc_nand_enable function

Name	Description
Function name	xmc_nand_enable
Function prototype	void xmc_nand_enable(xmc_class_bank_type xmc_bank, confirm_state new_state);
Function description	Enable nand flash controller
Input parameter 1	xmc_bank: selected xmc bank
Input parameter 2	new_state: enable or disable controller
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

xmc_bank

Select an XMC bank

XMC_BANK2_NAND

XMC_BANK3_NAND

new_state

FALSE: Disable controller

TRUE: Enable controller

Example:

```
/* xmc nand bank enable*/
xmc_nand_enable(XMC_BANK2_NAND, TRUE);
```

5.27.14 xmc_nand_ecc_enable function

The table below describes the function xmc_nand_ecc_enable.

Table 732. xmc_nand_ecc_enable function

Name	Description
Function name	xmc_nand_ecc_enable
Function prototype	void xmc_nand_ecc_enable(xmc_class_bank_type xmc_bank, confirm_state new_state);
Function description	Enable nand flash controller ECC feature
Input parameter 1	xmc_bank: selected xmc bank
Input parameter 2	new_state: enable or disable controller ECC feature
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

xmc_bank

Select an XMC bank

XMC_BANK2_NAND

XMC_BANK3_NAND

new_state

FALSE: Disable controller ECC feature

TRUE: Enable controller ECC feature

Example:

```
/* calculate ecc value while transmiting */
xmc_nand_ecc_enable(XMC_BANK2_NAND, TRUE);
```

5.27.15 xmc_ecc_get function

The table below describes the function xmc_ecc_get.

Table 733. xmc_ecc_get function

Name	Description
Function name	xmc_ecc_get
Function prototype	uint32_t xmc_ecc_get(xmc_class_bank_type xmc_bank);
Function description	Get nand flash controller ECC value
Input parameter 1	xmc_bank: selected xmc bank
Output parameter	NA
Return value	ECC value
Required preconditions	NA
Called functions	NA

xmc_bank

Select an XMC bank

XMC_BANK2_NAND

XMC_BANK3_NAND

Example:

```
/* get ecc value */
xmc_ecc_get(XMC_BANK2_NAND, TRUE);
```

5.27.16 xmc_interrupt_enable function

The table below describes the function xmc_interrupt_enable.

Table 734. xmc_interrupt_enable function

Name	Description
Function name	xmc_interrupt_enable
Function prototype	void xmc_interrupt_enable(xmc_class_bank_type xmc_bank, xmc_interrupt_sources_type xmc_int, confirm_state new_state);
Function description	Enable xmc controller interrupts
Input parameter 1	xmc_bank: selected xmc bank
Input parameter 2	xmc_int: interrupt selection
Input parameter 3	new_state: enable or disable interrupt
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

xmc_bank

Select a bank

XMC_BANK2_NAND

XMC_BANK3_NAND

XMC_BANK4_PCCARD

XMC_BANK5_6_SDRAM

xmc_int

XMC_INT_RISING_EDGE: XMC rising edge detection interrupt

XMC_INT_LEVEL: XMC high level detection interrupt

XMC_INT_FALLING_EDGE: XMC falling edge detection interrupt

XMC_INT_ERR: XMC error interrupt

new_state

FALSE: Disable interrupt

TRUE: Enable interrupt

Example:

```
/* xmc rising edge detection interrupt enable */
xmc_interrupt_enable(XMC_BANK2_NAND, XMC_INT_RISING_EDGE, TRUE);
```

5.27.17 xmc_flag_status_get function

The table below describes the function xmc_flag_status_get.

Table 735. xmc_flag_status_get function

Name	Description
Function name	xmc_flag_status_get
Function prototype	flag_status xmc_flag_status_get(xmc_class_bank_type xmc_bank, xmc_interrupt_flag_type xmc_flag);
Function description	Get XMC flag
Input parameter 1	xmc_bank: selected xmc bank

Name	Description
Input parameter 2	xmc_flag: flag selection
Output parameter	NA
Return value	flag_status: indicates whether the flag is set or not
Required preconditions	NA
Called functions	NA

xmc_bank

select a bank

XMC_BANK2_NAND

XMC_BANK3_NAND

XMC_BANK4_PCCARD

XMC_BANK5_6_SDRAM

xmc_flag

xmc_flag is used to select a flag, including:

XMC_RISINGEDGE_FLAG: Rising edge

XMC_LEVEL_FLAG: High level

XMC_FALLINGEDGE_FLAG: Falling edge

XMC_FEMPT_FLAG: FIFO empty

XMC_BUSY_FLAG: Busy

XMC_ERR_FLAG: Error

flag_status

RESET: The corresponding flag is reset

SET: The corresponding flag is set

Example:

```
if(xmc_flag_status_get (XMC_BANK2_NAND, XMC_RISINGEDGE_FLAG) != RESET)
{
  ……
}
```

5.27.18 xmc_interrupt_flag_status_get function

The table below describes the function xmc_flag_clear.

Table 736. xmc_flag_clear function

Name	Description
Function name	xmc_interrupt_flag_status_get
Function prototype	flag_status xmc_interrupt_flag_status_get(xmc_class_bank_type xmc_bank, xmc_interrupt_flag_type xmc_flag)
Function description	Get XMC interrupt flag status
Input parameter 1	xmc_bank: select xmc bank
Input parameter 2	xmc_flag:
Output parameter	NA
Return value	flag_status: SET or RESET
Required preconditions	NA
Called functions	NA

xmc_bank

Select a bank

XMC_BANK2_NAND

XMC_BANK3_NAND

XMC_BANK4_PCCARD

XMC_BANK5_6_SDRAM

xmc_flag

xmc_flag is used to select a flag, including:

XMC_RISINGEDGE_FLAG: Rising edge

XMC_LEVEL_FLAG: High level

XMC_FALLINGEDGE_FLAG: Falling edge

flag_status

RESET: The corresponding flag is reset

SET: The corresponding flag is set

Example:

```
if(xmc_interrupt_flag_status_get (XMC_BANK2_NAND, XMC_RISINGEDGE_FLAG) != RESET)
{
  ……
}
```

5.27.19 xmc_flag_clear function

The table below describes the function xmc_flag_clear.

Table 737. xmc_flag_clear function

Name	Description
Function name	xmc_flag_clear
Function prototype	void xmc_flag_clear(xmc_class_bank_type xmc_bank, xmc_interrupt_flag_type xmc_flag);
Function description	Clear flag
Input parameter 1	xmc_bank: selected xmc bank
Input parameter 2	xmc_flag: to-be-cleared flag
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

xmc_bank

Select a bank

XMC_BANK2_NAND

XMC_BANK3_NAND

XMC_BANK4_PCCARD

XMC_BANK5_6_SDRAM

xmc_flag

xmc_flag is used to select a flag, including:

XMC_RISINGEDGE_FLAG: Rising edge

XMC_LEVEL_FLAG: High level

XMC_FALLINGEDGE_FLAG: Falling edge

XMC_FEMPT_FLAG: FIFO empty

XMC_BUSY_FLAG: Busy

XMC_ERR_FLAG: Error

Example:

```
if(xmc_flag_status_get (XMC_BANK2_NAND, XMC_RISINGEDGE_FLAG) != RESET)
{
  ……
  xmc_flag_clear(XMC_BANK2_NAND, XMC_RISINGEDGE_FLAG);
}
```

5.27.20 xmc_pccard_reset function

The table below describes the function xmc_pccard_reset.

Table 738. xmc_pccard_reset function

Name	Description
Function name	xmc_pccard_reset
Function prototype	void xmc_pccard_reset(void);
Function description	Reset pccard controller
Input parameter	NA

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

Example:

```
/* reset pccard */
xmc_pccard_reset();
```

5.27.21 xmc_pccard_init function

The table below describes the function xmc_pccard_init.

Table 739. xmc_pccard_init function

Name	Description
Function name	xmc_pccard_init
Function prototype	void xmc_pccard_init(xmc_pccard_init_type* xmc_pccard_init_struct);
Function description	Initialize pccard controller
Input parameter 1	xmc_pccard_init_struct: xmc_pccard_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

xmc_pccard_init_type struct

The xmc_pccard_init_type is defined in the at32f435_437_xmc.h.

```
typedef struct
{
  xmc_nand_pccard_wait_type        enable_wait;
  uint32_t                         delay_time_cr;
  uint32_t                         delay_time_ar;
} xmc_pccard_init_type;
```

enable_wait

Wait enable bit. It is used to enable PC card memory block wait feature.

XMC_WAIT_OPERATION_DISABLE: Disable

XMC_WAIT_OPERATION_ENABLE: Enable

delay_time_cr

Delay time of CLE to RE (from CLE falling edge to RE falling edge)

0000: 1 HCLKclock cycle

……

1111: 16 HCLK clock cycles

delay_time_ar

Delay time of ALE to RE (from ALE falling edge to RE falling edge)

0000: 1 HCLK clock cycle

……

1111: 16 HCLK clock cycles

Example:

```
xmc_pccard_init_type xmc_pccard_init_struct;
xmc_pccard_init_struct. enable_wait = XMC_WAIT_OPERATION_DISABLE;
xmc_pccard_init_struct. delay_time_cr = 0xF;
xmc_pccard_init_struct. delay_time_ar = 0xF;
/* xmc pccard configuration */
xmc_pccard_init (&xmc_pccard_init_struct);
```

5.27.22 xmc_pccard_timing_config function

The table below describes the function xmc_pccard_timing_config.

Table 740. xmc_nand_pccard_config function

Name	Description
Function name	xmc_pccard_timing_config
Function prototype	void xmc_pccard_timing_config(xmc_nand_pccard_timinginit_type* xmc_common_spacetiming_struct, xmc_nand_pccard_timinginit_type* xmc_attribute_spacetiming_struct, xmc_nand_pccard_timinginit_type* xmc_iospace_timing_struct);
Function description	Configure pccard controller timing
Input parameter 1	xmc_common_spacetiming_struct: xmc_nand _pccard_timinginit_type pointer It indicates regular memory timing parameters.
Input parameter 2	xmc_ attribute _spacetiming_struct: xmc_nand _pccard_timinginit_type pointer It indicates special memory timing parameters.
Input parameter 3	xmc_iospace_timing_struct: xmc_nand_pccard_timinginit_type pointer It indicates special memory timing parameters.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

xmc_nand_pccard_timinginit_type struct

The xmc_nand_pccard_timinginit_type is defined in the at32f435_437_xmc.h.

```
typedef struct
{
  xmc_class_bank_type      class_bank;
  uint32_t                 mem_setup_time;
  uint32_t                 mem_waite_time;
  uint32_t                 mem_hold_time;
  uint32_t                 mem_hiz_time;
} xmc_nand_pccard_timinginit_type;
```

class_bank

NAND flash bank selection

XMC_BANK4_PCCARD

mem_setup_time

Regular memory setup time. It defines the address setup time during access operation in a regular memory.

00000000: 0 HCLK clock cycle is inserted

00000001: 1 HCLK clock cycle is inserted

……

11111111: 255 HCLK clock cycles are inserted

mem_waite_time

Regulary memory wait time. It defines the regular memory wait time when the XMC_NWE and XMC_NOE are low.

00000000: 0 HCLK clock cycle is inserted

00000001: 1 HCLK clock cycle is inserted

……

11111111: 255 HCLK clock cycles are inserted

mem_hold_time

Regular memory hold time. It defines the databus hold time during access operation in a regular memory.

00000000: Reserved

00000001: 1 HCLK clock cycle is inserted

……

11111111: 255 HCLK clock cycles are inserted

mem_hiz_time

Regular memory databus High resistance time. It defines the databus high resistance duration when write access to PC card is started in a regular space.

00000000: 0 HCLK clock cycle is inserted

00000001: 1 HCLK clock cycle is inserted

……

11111111: 255 HCLK clock cycles are inserted

Example:

```
xmc_nand_pccard_timinginit_type xmc_common_spacetiming_struct, xmc_attribute_spacetiming_struct,
xmc_iospace_timing_struct;
xmc_common_spacetiming_struct.class_bank = XMC_BANK4_PCCARD;
xmc_common_spacetiming_struct.mem_setup_time = 254;
xmc_common_spacetiming_struct.mem_hiz_time = 254;
xmc_common_spacetiming_struct.mem_hold_time = 254;
xmc_common_spacetiming_struct.mem_waite_time = 254;

xmc_attribute_spacetiming_struct.class_bank = XMC_BANK4_PCCARD;
xmc_attribute_spacetiming_struct.mem_setup_time = 254;
xmc_attribute_spacetiming_struct.mem_hiz_time = 254;
xmc_attribute_spacetiming_struct.mem_hold_time = 254;
xmc_attribute_spacetiming_struct.mem_waite_time = 254;

xmc_iospace_timing_struct.class_bank = XMC_BANK4_PCCARD;
xmc_iospace_timing_struct.mem_setup_time = 254;
xmc_iospace_timing_struct.mem_hiz_time = 254;
xmc_iospace_timing_struct.mem_hold_time = 254;
xmc_iospace_timing_struct.mem_waite_time = 254;

xmc_pccard_timing_config (&xmc_common_spacetiming_struct, & xmc_attribute_spacetiming_struct,&
xmc_iospace_timing_struct);
```

5.27.23 xmc_pccard_default_para_init function

The table below describes the function xmc_pccard_default_para_init.

Table 741. xmc_pccard_default_para_init function

Name	Description
Function name	xmc_pccard_default_para_init
Function prototype	void xmc_pccard_default_para_init(xmc_pccard_init_type* xmc_pccard_init_struct);
Function description	Initialize parameters in xmc_pccard_init_struct
Input parameter 1	xmc_pccard_init_struct: xmc_pccard_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of members xmc_nand_init_struct.

Table 742.xmc_pccard_init_struct default values

Member	Default value
enable_wait	XMC_WAIT_OPERATION_DISABLE
delay_time_ar	0x0
delay_time_cr	0x0

Example:

```
/* fill each xmc_pccard_init_struct member with its default value */
xmc_pccard_default_para_init (&xmc_pccard_init_struct);
```

5.27.24 xmc_pccard_timing_default_para_init function

The table below describes the function xmc_pccard_timing_default_para_init.

Table 743. xmc_pccard_timing_default_para_init function

Name	Description
Function name	xmc_pccard_timing_default_para_init
Function prototype	void xmc_pccard_timing_default_para_init(xmc_nand_pccard_timinginit_type* xmc_common_spacetiming_struct, xmc_nand_pccard_timinginit_type* xmc_attribute_spacetiming_struct, xmc_nand_pccard_timinginit_type* xmc_iospace_timing_struct);
Function description	Initialize parameters in xmc_common_spacetiming_struct, xmc_attribute_spacetiming_struct and xmc_iospace_timing_struct
Input parameter 1	xmc_common_spacetiming_struct: xmc_nand_pccard_timinginit_type pointer
Input parameter 2	xmc_attribute_spacetiming_struct: xmc_nand_pccard_timinginit_type pointer
Input parameter 3	xmc_iospace_timing_struct: xmc_nand_pccard_timinginit_type pointer
Output parameter	NA
Return value	NA
Required	NA

Name	Description
preconditions	
Called functions	NA

The table below describes the default values of members xmc_common_spacetiming_struct, xmc_attribute_spacetiming_struct and xmc_iospace_timing_struct.

Table 744.xmc_common_spacetiming_struct and xmc_attribute_spacetiming_struct default values

Member	Default value
class_bank	XMC_BANK4_PCCARD
mem_hold_time	0xFC
mem_waite_time	0xFC
mem_setup_time	0xFC
mem_hiz_time	0xFC

Example:

```
xmc_pccard_timinginit_type  xmc_regular_spacetimingstruct;
/* fill each xmc_regular_spacetimingstruct member with its default value */
xmc_pccard_timing_default_para_init (&xmc_regular_spacetimingstruct, &xmc_regular_spacetimingstruct，&
xmc_regular_spacetimingstruct);
```

5.27.25 xmc_pccard_enable function

The table below describes the function xmc_pccard_enable.

Table 745. xmc_pccard_enable function

Name	Description
Function name	xmc_pccard_enable
Function prototype	void xmc_pccard_enable(confirm_state new_state);
Function description	Enable pccard controller
Input parameter 1	new_state: enable or disable
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

FALSE: Disable controller

TRUE: Enable controller

Example:

```
/* xmc pccard bank enable*/
xmc_pccard_enable(TRUE);
```

5.27.26 xmc_sdram_reset function

The table below describes the function xmc_sdram_reset.

Table 746. xmc_sdram_reset function

Name	Description
Function name	xmc_sdram_reset
Function prototype	void xmc_sdram_reset(xmc_sdram_bank_type xmc_bank);
Function description	Reset sdram bank controller
Input parameter 1	xmc_bank: selected sdram bank
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

xmc_bank

XMC_SDRAM_BANK1: SDRAM bank1

XMC_SDRAM_BANK2: SDRAM bank2

Example:

```
/* xmc sdram bank reset*/
xmc_sdram_reset(XMC_SDRAM_BANK1);
```

5.27.27 xmc_sdram_init function

The table below describes the function xmc_sdram_init.

Table 747. xmc_sdram_init function

Name	Description
Function name	xmc_sdram_init
Function prototype	void xmc_sdram_init(xmc_sdram_init_type *xmc_sdram_init_struct, xmc_sdram_timing_type *xmc_sdram_timing_struct);
Function description	Initialize sdram bank
Input parameter 1	xmc_sdram_init_struct: xmc_sdram_init_type pointer
Input parameter 2	xmc_sdram_timing_struct: xmc_sdram_timing_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

xmc_sdram_init_struct

The xmc_sdram_init_type is defined in the at32f435_437_xmc.h.

```
typedef struct
{
  xmc_sdram_bank_type          sdram_bank;
  xmc_sdram_inbk_type          internel_banks;
  xmc_sdram_clkdiv_type        clkdiv;
  uint8_t                      write_protection;
  uint8_t                      burst_read;
```

```
  uint8_t                    read_delay;
  xmc_sdram_column_type      column_address;
  xmc_sdram_row_type         row_address;
  xmc_sdram_cas_type         cas;
  xmc_sdram_width_type       width;
} xmc_sdram_init_type;
```

sdram_bank

Select a sdram bank

XMC_SDRAM_BANK1: sdram bank1

XMC_SDRAM_BANK2: sdram bank2

internel_banks

number of banks supported by sdram device

XMC_INBK_2: 2 internal banks

XMC_INBK_4: 4 internal banks

clkdiv

Set sdram clock division

XMC_NO_CLK: sdram clock disabled

XMC_CLKDIV_2: Divided by 2

XMC_CLKDIV_3: Divided by 3

XMC_CLKDIV_4: Divided by 4

write_protection

Enable/disable sdram write protection

TRUE: Enable sdram write protection

FALSE: Disable sdram write protection

burst_read

Enable/disable sdram consecutive read operations

TRUE: Enable sdram consecutive read operations

FALSE: Disable sdram consecutive read operations

read_delay

Set read delay feature

XMC_READ_DELAY_0: 0 HCLK clock cycle delay

XMC_READ_DELAY_1: 1 HCLK clock cycle delay

XMC_READ_DELAY_2: 2 HCLK clock cycles delay

column_address

Number of column address

XMC_COLUMN_8: 8 bits

XMC_COLUMN_9: 9 bits

XMC_COLUMN_10: 10 bits

XMC_COLUMN_11: 11 bits

row_address

Number of row address

XMC_ROW_11: 11 bits

XMC_ROW_12: 12 bits

XMC_ROW_13: 13 bits

cas

CAS latency
XMC_CAS_1: 1 cycle
XMC_CAS_2: 2 cycles
XMC_CAS_3: 3 cycles

width
Data bus width
XMC_MEM_WIDTH_8: 8-bit
XMC_MEM_WIDTH_16: 16-bit

xmc_sdram_timing_struct
The xmc_sdram_timing_type is defined in the at32f435_437_xmc.h.

tmrd
Delay between a load mode register command and an activate or refresh command in number
XMC_DELAY_CYCLE_1: 1 cycle
XMC_DELAY_CYCLE_2: 2 cycles
……
XMC_DELAY_CYCLE_15: 15 cycles
XMC_DELAY_CYCLE_16: 16 cycles

txsr
Exit self-refresh to active delay
XMC_DELAY_CYCLE_1: 1 cycle
XMC_DELAY_CYCLE_2: 2 cycles
……
XMC_DELAY_CYCLE_15: 15 cycles
XMC_DELAY_CYCLE_16: 16 cycles

tras
Self-refresh time
XMC_DELAY_CYCLE_1: 1 cycle
XMC_DELAY_CYCLE_2: 2 cycles
……
XMC_DELAY_CYCLE_15: 15 cycles
XMC_DELAY_CYCLE_16: 16 cycles

trc
Refresh to active delay
XMC_DELAY_CYCLE_1: 1 cycle
XMC_DELAY_CYCLE_2: 2 cycles
……
XMC_DELAY_CYCLE_15: 15 cycles
XMC_DELAY_CYCLE_16: 16 cycles

twr
Delay between a write command and a precharge command
XMC_DELAY_CYCLE_1: 1 cycle
XMC_DELAY_CYCLE_2: 2 cycles
……
XMC_DELAY_CYCLE_15: 15 cycles
XMC_DELAY_CYCLE_16: 16 cycles

trp

Delay between a precharge command and active command

XMC_DELAY_CYCLE_1: 1 cycle

XMC_DELAY_CYCLE_2: 2 cycles

……

XMC_DELAY_CYCLE_15: 15 cycles

XMC_DELAY_CYCLE_16: 16 cycles

trcd

Delay between an active command and a write/read command

XMC_DELAY_CYCLE_1: 1 cycle

XMC_DELAY_CYCLE_2: 2 cycles

……

XMC_DELAY_CYCLE_15: 15 cycles

XMC_DELAY_CYCLE_16: 16 cycles

Example:

```
xmc_sdram_init_type sdram_init_struct;
xmc_sdram_timing_type sdram_timing_struct;
/*-- xmc configuration ----------------------------------------------------*/
xmc_sdram_default_para_init(&sdram_init_struct, &sdram_timing_struct);
sdram_init_struct.sdram_bank                = XMC_SDRAM_BANK1;
sdram_init_struct.internel_banks            = XMC_INBK_4;
sdram_init_struct.clkdiv                    = XMC_CLKDIV_3;
sdram_init_struct.write_protection          = FALSE;
sdram_init_struct.burst_read                = FALSE;
sdram_init_struct.read_delay                = XMC_READ_DELAY_1;
sdram_init_struct.column_address            = XMC_COLUMN_9;
sdram_init_struct.row_address               = XMC_ROW_13;
sdram_init_struct.cas                       = XMC_CAS_3;
sdram_init_struct.width                     = XMC_MEM_WIDTH_16;

sdram_timing_struct.tmrd                    = XMC_DELAY_CYCLE_2;
sdram_timing_struct.txsr                    = XMC_DELAY_CYCLE_11;
sdram_timing_struct.tras                    = XMC_DELAY_CYCLE_7;
sdram_timing_struct.trc                     = XMC_DELAY_CYCLE_9;
sdram_timing_struct.twr                     = XMC_DELAY_CYCLE_2;
sdram_timing_struct.trp                     = XMC_DELAY_CYCLE_3;
sdram_timing_struct.trcd                    = XMC_DELAY_CYCLE_3;
xmc_sdram_init(&sdram_init_struct, &sdram_timing_struct);
```

5.27.28 xmc_sdram_default_para_init function

The table below describes the function xmc_sdram_default_para_init.

Table 748. xmc_sdram_ default_para_init function

Name	Description
Function name	xmc_sdram_default_para_init
Function prototype	void xmc_sdram_default_para_init(xmc_sdram_init_type *xmc_sdram_init_struct, xmc_sdram_timing_type *xmc_sdram_timing_struct);
Function description	Initialize parameters in xmc_sdram_init_struct and xmc_sdram_timing_struct
Input parameter 1	sdram controller initialization parameters. Refer to *xmc_sdram_init_struct* for details.
Input parameter 2	sdram timing parameters Refer to *xmc_sdram_timing_struct* for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
xmc_sdram_init_type sdram_init_struct;
xmc_sdram_timing_type sdram_timing_struct;
xmc_sdram_default_para_init(&sdram_init_struct, &sdram_timing_struct);
```

5.27.29 xmc_sdram_cmd function

The table below describes the function xmc_sdram_cmd.

Table 749. xmc_sdram_cmd function

Name	Description
Function name	xmc_sdram_cmd
Function prototype	void xmc_sdram_cmd(xmc_sdram_cmd_type *xmc_sdram_cmd_struct);
Function description	Send sdram command
Input parameter 1	xmc_sdram_cmd_struct: xmc_sdram_cmd_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

xmc_sdram_cmd_struct

xmc_sdram_cmd_type is defined in the at32f435_437_xmc.h.

cmd

Command sent to sdram device

XMC_CMD_NORMAL: Normal mode
XMC_CMD_CLK: Clock configuration enable
XMC_CMD_PRECHARG_ALL: Precharge all banks
XMC_CMD_AUTO_REFRESH: : Auto refresh
XMC_CMD_LOAD_MODE: Load mode register

XMC_CMD_SELF_REFRESH: Self-refresh
XMC_CMD_POWER_DOWN: Power-down command

cmd_banks

Select a sdram bank
XMC_CMD_BANK1: Command sent to sdram bank1
XMC_CMD_BANK2: Command sent to sdram bank1
XMC_CMD_BANK1_2: Command sent to sdram bank1 and bank2

auto_refresh

Auto-refresh times
0: 1 auto-refresh cycle
1: 2 auto-refresh cycles
……
14: 15 auto-refresh cycles

data

Mode register data

Example:

```
xmc_sdram_cmd_type sdram_cmd_struct;
sdram_cmd_struct.cmd                = XMC_CMD_CLK;
sdram_cmd_struct.auto_refresh       = 1;
sdram_cmd_struct.cmd_banks          = XMC_CMD_BANK1;
sdram_cmd_struct.data               = 0;
xmc_sdram_cmd(&sdram_cmd_struct);
```

5.27.30 xmc_sdram_status_get function

The table below describes the function xmc_sdram_status_get.

Table 750. xmc_sdram_status_get function

Name	Description
Function name	xmc_sdram_status_get
Function prototype	uint32_t xmc_sdram_status_get(xmc_sdram_bank_type xmc_bank);
Function description	Get sdram bank status
Input parameter 1	xmc_bank: selected sdram bank
Output parameter	NA
Return value	Return the current status. Refer to *xmc_bank_status_type* for details.
Required preconditions	NA
Called functions	NA

xmc_bank

Select a sdram bank
XMC_SDRAM_BANK1: sdram bank1
XMC_SDRAM_BANK2: sdram bank2

xmc_bank_status_type

XMC_STATUS_NORMAL: normal mode
XMC_STATUS_SELF_REFRESH: seld-refresh mode

XMC_STATUS_POWER_DOWN: power-down mode

Example:

```
/* get sdram status */
xmc_sdram_status_get(XMC_SDRAM_BANK1)
```

5.27.31 xmc_sdram_refresh_counter_set function

The table below describes the function xmc_sdram_refresh_counter_set.

Table 751. xmc_sdram_refresh_counter_set function

Name	Description
Function name	xmc_sdram_refresh_counter_set
Function prototype	void xmc_sdram_refresh_counter_set(uint32_t counter);
Function description	Set sdram auto-refresh counter
Input parameter 1	counter: auto-refresh counter value
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/*set refresh counter */
xmc_sdram_refresh_counter_set(1105);
```

5.27.32 xmc_sdram_auto_refresh_set function

The table below describes the function xmc_sdram_auto_refresh_set.

Table 752. xmc_sdram_auto_refresh_set function

Name	Description
Function name	xmc_sdram_auto_refresh_set
Function prototype	void xmc_sdram_auto_refresh_set(uint32_t number)
Function description	Set sdram software auto-refresh times
Input parameter 1	number: auto-refresh times
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set 8 times auto refresh */
xmc_sdram_auto_refresh_set(8);
```

6 Precautions

6.1 Device model replacement

While replacing the device part number in an existing project or demo with another one, 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

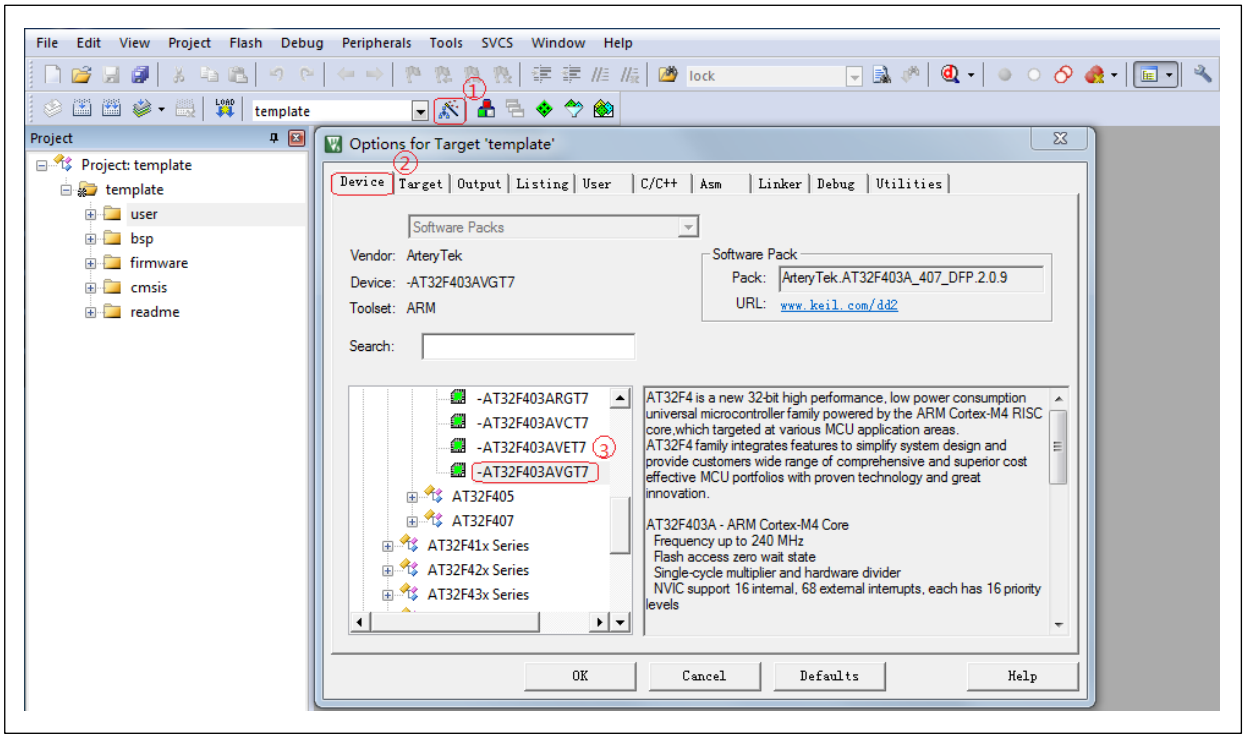
Follow the steps and illustration below for device replacement in Keil environment.

① Click on magic wand “Options for Target”

② Click on “Device”

③ Select the desired device part number

Figure 29. Change device part number in Keil



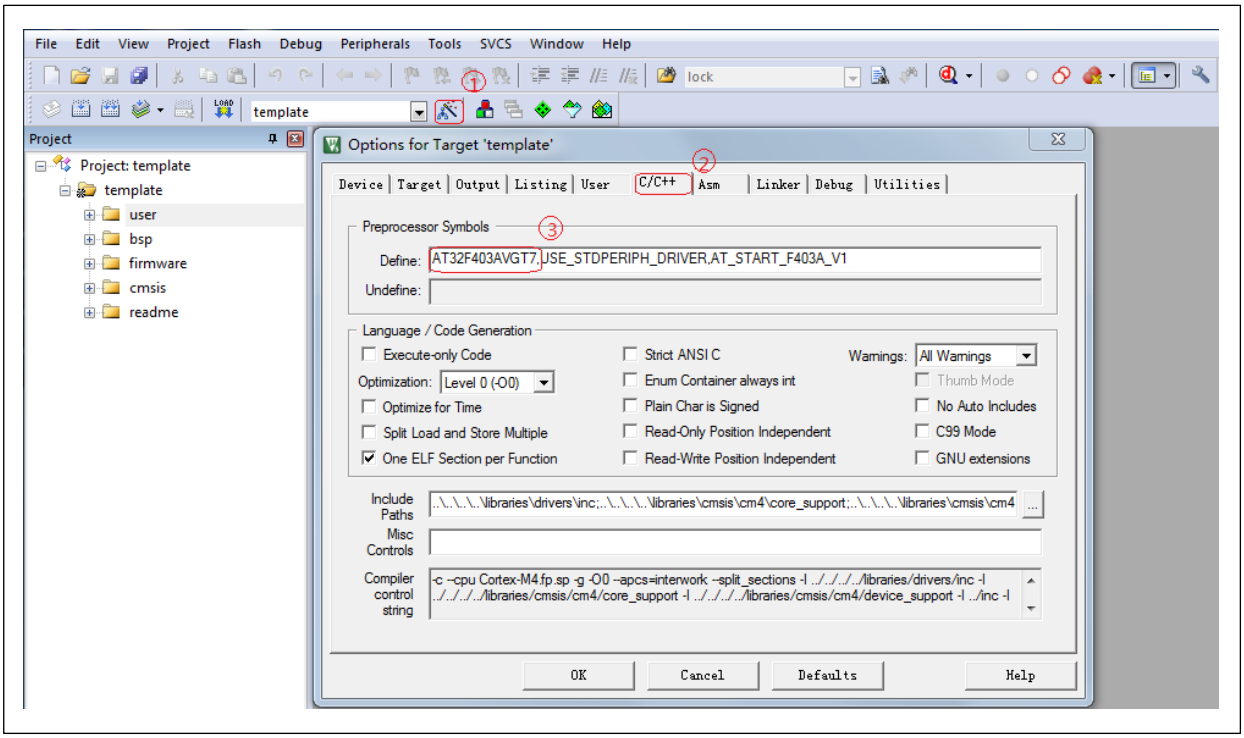
Follow the steps and illustration below to change macro definition.

① Click on magic wand “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 on Table 1.

Figure 30. Change macro definition in Keil



6.1.2 IAR environment

Follow the steps and illustration below for device replacement in IAR environment

① Right click on the file name, and select “Options…”

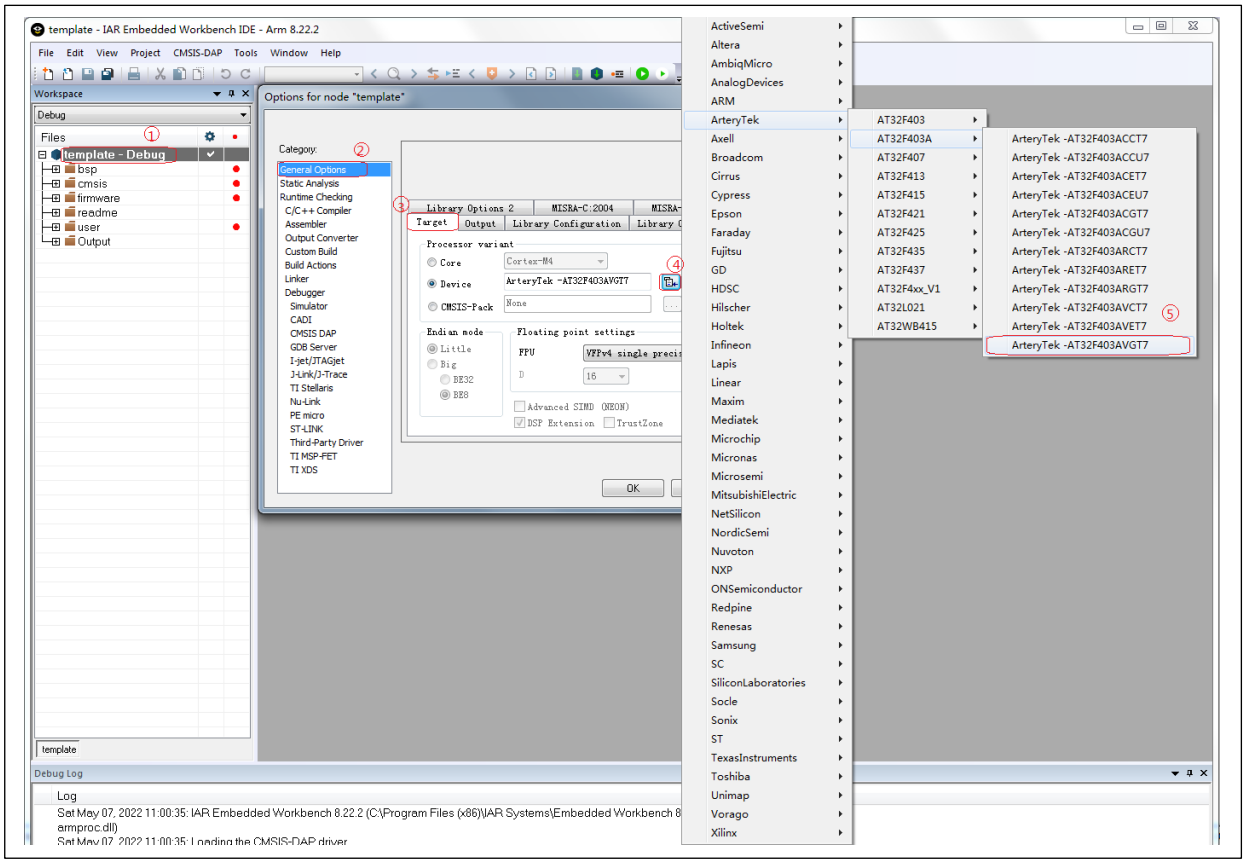
② Select “General Options”

③ Select “Target”

④ Click on the 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.

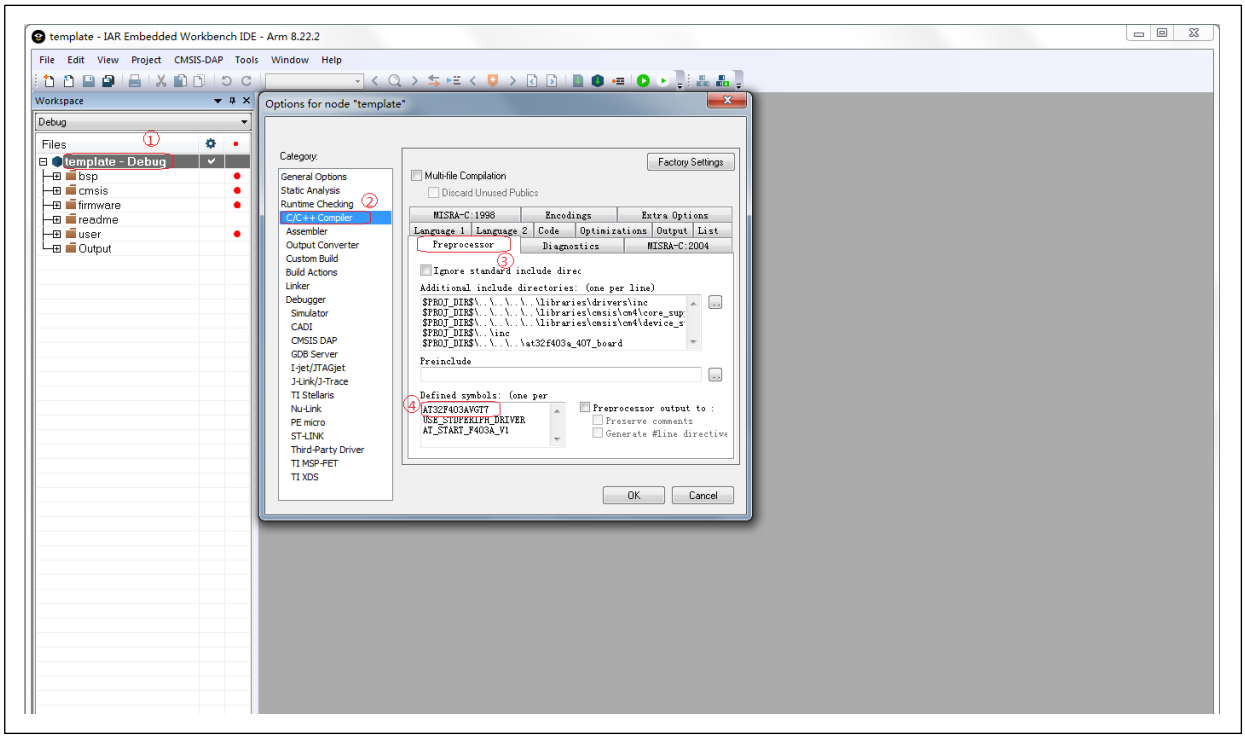
① Right click on the file name, and select “Options…”

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

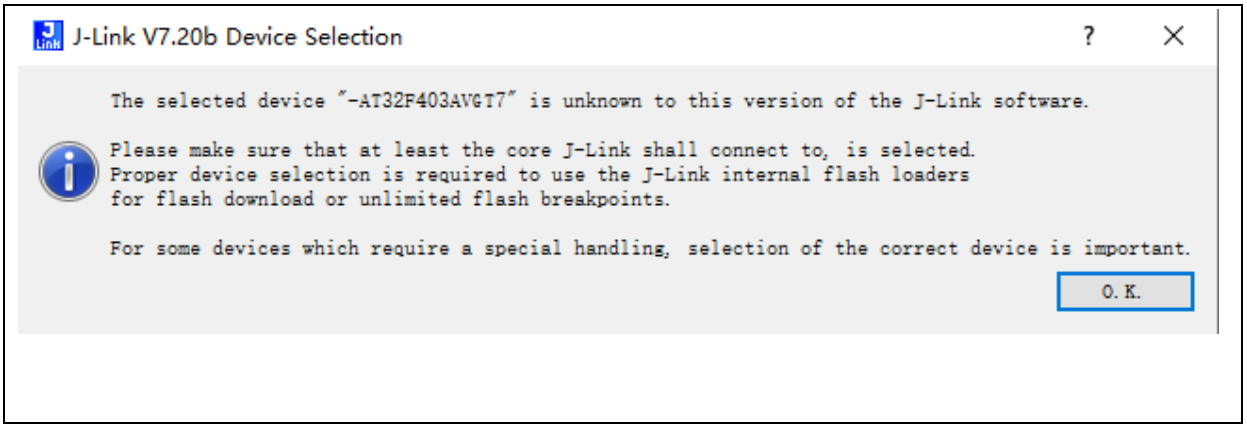


Figure 34. Error warning 2

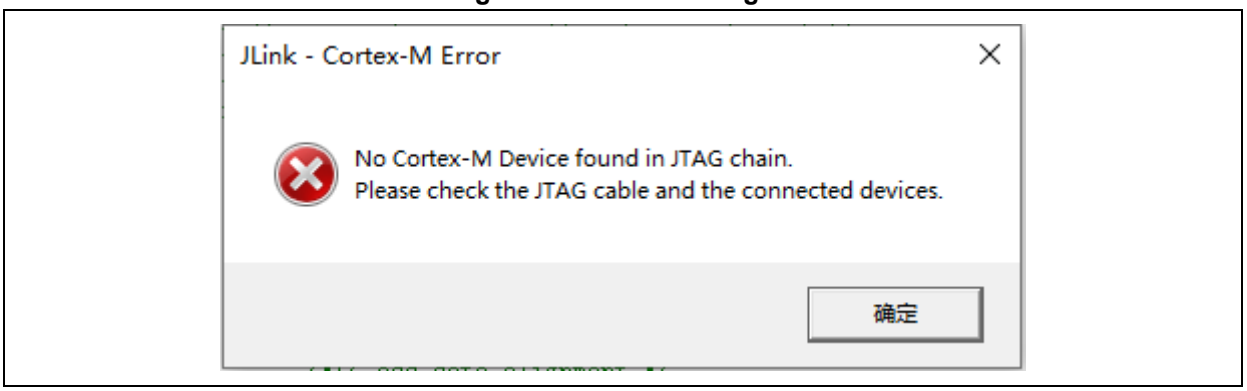


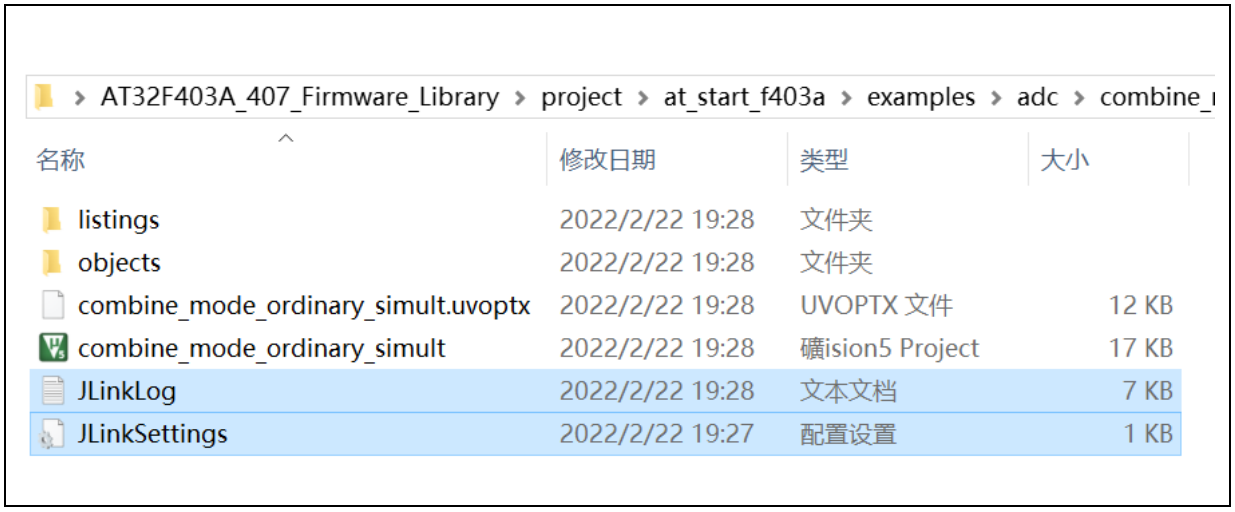
Figure 35. Error warning 3



How to solve this problem?

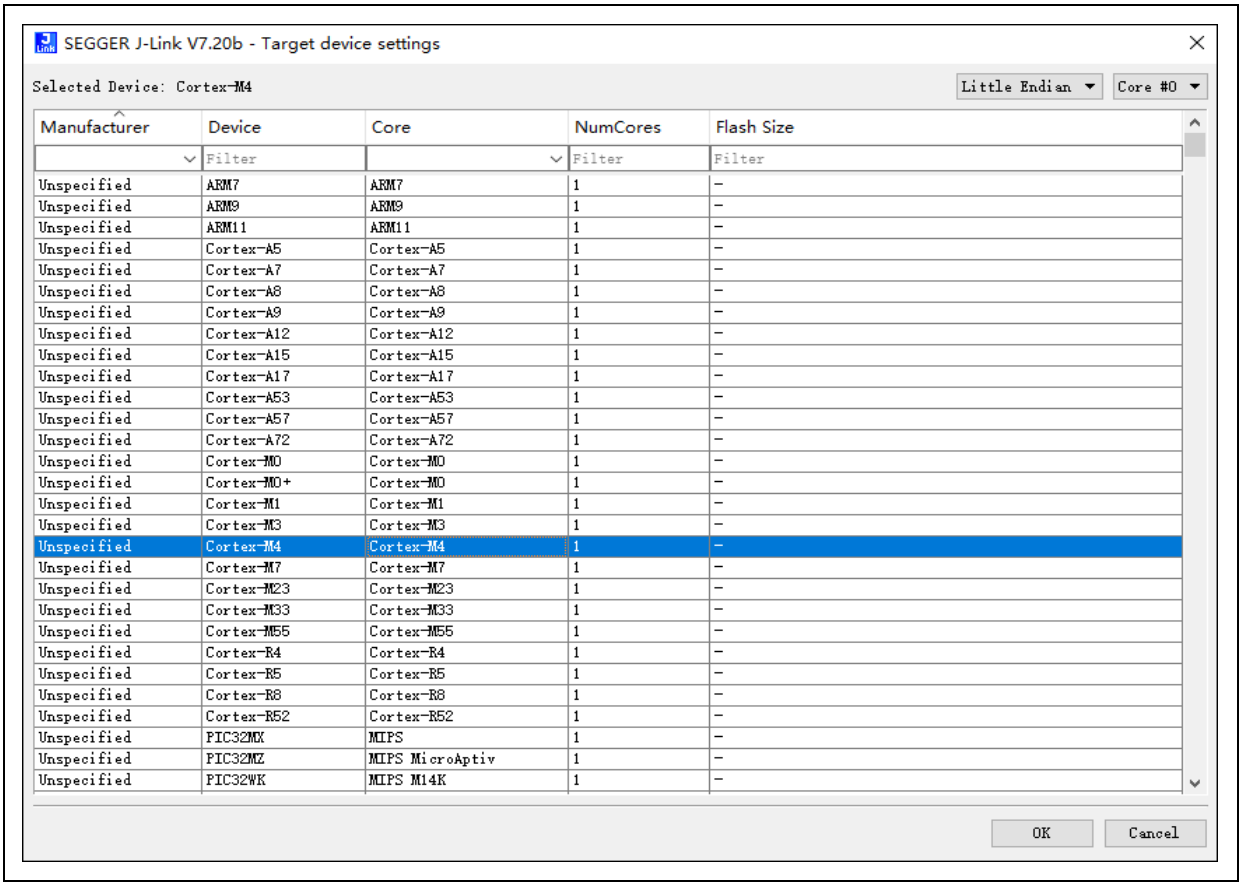
Step 1: Find “JLinkLog” and “JLinkSettings” files according to project path, and delete them

Figure 36. JLinkLog and JLinkSettings



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

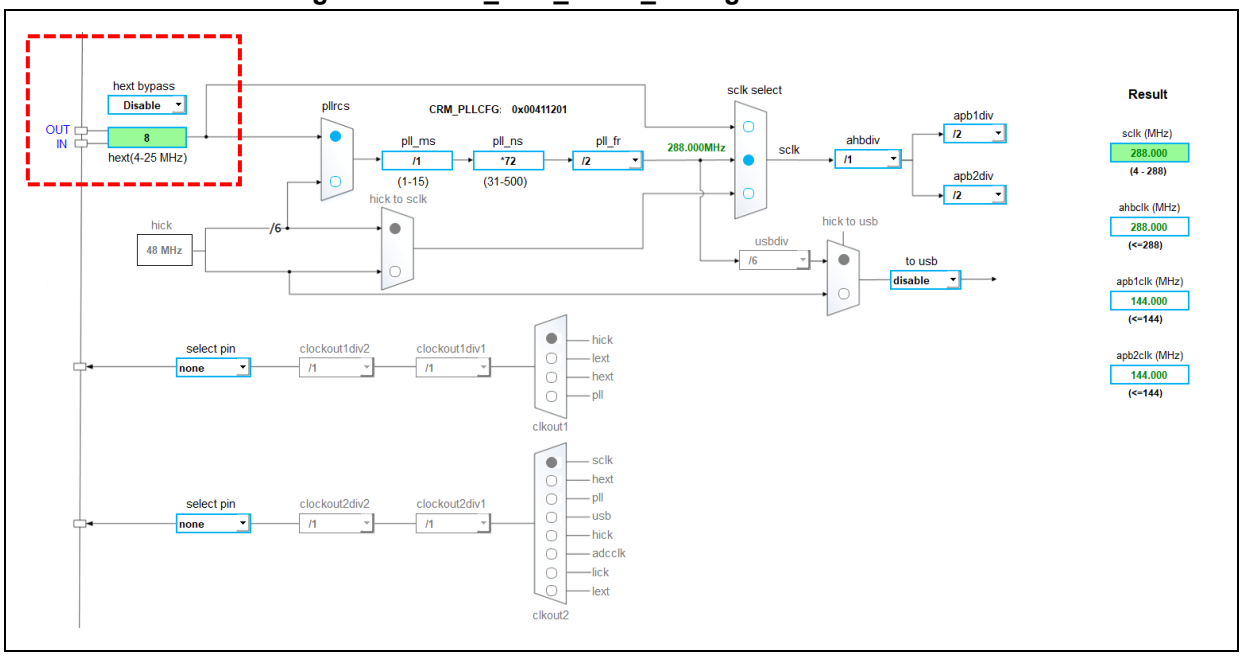
All examples used in BSP implements frequency multiplication based on 8 MHz external highspeed crystal oscillator on the evaluation board. If a non-8 MHz external crystal is used in actual scenarios, it is necessary to modify clock configuration in BSP to allow for accurate and stable clock frequency.

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 exernal 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 the table below, which are all available from the official website of Artery.

Table 753. 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 754. Document revision history

Date	Version	Revision note
2021.09.06	2.0.0	Initial release
2021.11.19	2.0.1	Updated screenshots and descriptions in the Pack installation section
2022.05.09	2.0.2	Added Section 6 Precautions
2022.06.15	2.0.3	Added descriptions of peripheral library functions
2022.11.15	2.0.4	Modified descriptions of I2C in “abbreviations of peripherals”
2023.07.18	2.0.5	Added section 5.4.10 to 5.4.13
2023.10.26	2.0.6	Added the function “interrupt_flag_get” to each of the sections of this file.
2024.01.04	2.0.7	Added the *dvp_interrupt_flag_get function*
2024.03.01	2.0.8	Added the *5.17.10 pwc_voltage_regulate_set function.* Changed the typo “PWC” to “NVIC” in section 5.16
2024.10.23	2.0.9	Removed ‘PWC_LDO_OUTPUT_1V0’ from ‘val’ in *Section 5.17.12pwc_ldo_output_voltage_set function*
2025.11.24	2.1.0	Added Section *5.14.11 gpio_bits_toggle function*

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.

© 2025 Artery Technology -All rights reserved