

AT32L021 Firmware BSP&Pack

Introduction

This application note is written to give a brief description of how to use AT32L021 BSP (Board Support Package) and install AT32 pack.

Contents

1	Overview	35
2	How to install Pack	36
2.1	IAR Pack installation.....	36
2.2	Keil_v5 Pack installation	38
2.3	Keil_v4 Pack installation	40
2.4	Segger Pack installation	42
3	Flash algorithm file	46
3.1	How to use Keil algorithm file	46
3.2	How to use IAR algorithm files.....	48
4	BSP introduction.....	52
4.1	Quick start	52
4.1.1	Template project.....	52
4.1.2	BSP macro definitions	53
4.2	BSP specifications.....	55
4.2.1	List of abbreviations for peripherals	55
4.2.2	Naming rules	55
4.2.3	Encoding rules.....	56
4.3	BSP structure	58
4.3.1	BSP folder structure	58
4.3.2	BSP function library structure.....	59
4.3.3	Initialization and configuration for peripherals	61
4.3.4	Peripheral functions format description.....	62
5	AT32L021 peripheral library functions	63
5.1	Controller area network (CAN)	63
5.1.1	can_reset function	65
5.1.2	can_baudrate_default_para_init function.....	65
5.1.3	can_baudrate_set function	66
5.1.4	can_default_para_init function	67

5.1.5	can_base_init function	67
5.1.6	can_filter_default_para_init function	69
5.1.7	can_filter_init function	69
5.1.8	can_debug_transmission_prohibit function	71
5.1.9	can_ttc_mode_enable function	71
5.1.10	can_message_transmit function	72
5.1.11	can_transmit_status_get function	74
5.1.12	can_transmit_cancel function	75
5.1.13	can_message_receive function	75
5.1.14	can_receive_fifo_release function	77
5.1.15	can_receive_message_pending_get function	78
5.1.16	can_operating_mode_set function.....	78
5.1.17	can_doze_mode_enter function.....	79
5.1.18	can_doze_mode_exit function	80
5.1.19	can_error_type_record_get function	80
5.1.20	can_receive_error_counter_get function	81
5.1.21	can_transmit_error_counter_get function	81
5.1.22	can_interrupt_enable function.....	82
5.1.23	can_flag_get function	83
5.1.24	can_interrupt_flag_get function.....	84
5.1.25	can_flag_clear function	85
5.2	CRC calculation unit (CRC).....	86
5.2.1	crc_data_reset function.....	87
5.2.2	crc_one_word_calculate function.....	87
5.2.3	crc_block_calculate function	88
5.2.4	crc_data_get function.....	88
5.2.5	crc_common_data_set function	89
5.2.6	crc_common_data_get function.....	89
5.2.7	crc_init_data_set function	90
5.2.8	crc_reverse_input_data_set function.....	90
5.2.9	crc_reverse_output_data_set function.....	91
5.2.10	crc_poly_value _set function.....	91
5.2.11	crc_poly_value _get function.....	92
5.2.12	crc_poly_size _set function	92
5.2.13	crc_poly_size _get function.....	93

5.3	Clock and reset management (CRM)	94
5.3.1	crm_reset function.....	96
5.3.2	crm_lxt_bypass function.....	96
5.3.3	crm_hext_bypass function	97
5.3.4	crm_flag_get function.....	97
5.3.5	crm_interrupt_flag_get function	98
5.3.6	crm_hext_stable_wait function.....	99
5.3.7	crm_hick_clock_trimming_set function	99
5.3.8	crm_hick_clock_calibration_set function	100
5.3.9	crm_periph_clock_enable	100
5.3.10	crm_periph_reset function.....	101
5.3.11	crm_periph_sleep_mode_clock_enable function	101
5.3.12	crm_clock_source_enable function.....	102
5.3.13	crm_flag_clear function	103
5.3.14	crm_ertc_clock_select function.....	104
5.3.15	crm_ertc_clock_enable function	105
5.3.16	crm_ahb_div_set function	105
5.3.17	crm_apb1_div_set function	106
5.3.18	crm_apb2_div_set function	106
5.3.19	crm_adc_clock_div_set function.....	107
5.3.20	crm_clock_failure_detection_enable function.....	107
5.3.21	crm_battery_powered_domain_reset function.....	108
5.3.22	crm_pll_config function	108
5.3.23	crm_pll_config2 function	109
5.3.24	crm_sysclk_switch function.....	110
5.3.25	crm_sysclk_switch_status_get function.....	110
5.3.26	crm_clocks_freq_get function	111
5.3.27	crm_clock_out_set function.....	112
5.3.28	crm_interrupt_enable function	112
5.3.29	crm_hick_sclk_frequency_select function	113
5.3.30	crm_clkout_div_set function	114
5.3.31	crm_usart_clock_select function.....	114
5.3.32	crm_usart_clock_get function	115
5.3.33	crm_crm_i2c_clock_select function	116
5.3.34	crm_crm_i2c_clock_get function.....	117

5.4	Debug.....	118
5.4.1	debug_device_id_get function	118
5.4.2	debug_device_series_id_get function.....	119
5.4.3	debug_periph _mode_set function.....	120
5.5	DMA controller	121
5.5.1	dma_default_para_init function	122
5.5.2	dma_init function	123
5.5.3	dma_reset function.....	125
5.5.4	dma_data_number_set function	125
5.5.5	dma_data_number_get function	126
5.5.6	dma_interrupt_enable function	126
5.5.7	dma_channel_enable function	127
5.5.8	dma_flexible_config function.....	127
5.5.9	dma_flag_get function.....	129
5.5.10	dma_interrupt_get function	130
5.5.11	dma_flag_clear function	131
5.6	Real-time clock (ERTC).....	132
5.6.1	ertc_num_to_bcd function.....	135
5.6.2	ertc_bcd_to_num function.....	135
5.6.3	ertc_write_protect_enable function	136
5.6.4	ertc_write_protect_disable function	136
5.6.5	ertc_wait_update function	137
5.6.6	ertc_wait_flag function	137
5.6.7	ertc_init_mode_enter function.....	138
5.6.8	ertc_init_mode_exit function	138
5.6.9	ertc_reset function.....	139
5.6.10	ertc_divider_set function	139
5.6.11	ertc_hour_mode_set function	140
5.6.12	ertc_date_set function.....	140
5.6.13	ertc_time_set function	141
5.6.14	ertc_calendar_get function.....	142
5.6.15	ertc_sub_second_get function	143
5.6.16	ertc_alarm_mask_set function	143
5.6.17	ertc_alarm_week_date_select function	144
5.6.18	ertc_alarm_set function.....	145

5.6.19	ertc_alarm_sub_second_set function	146
5.6.20	ertc_alarm_enable function.....	147
5.6.21	ertc_alarm_get function.....	147
5.6.22	ertc_alarm_sub_second_get function	149
5.6.23	ertc_wakeup_clock_set function	149
5.6.24	ertc_wakeup_counter_set function	150
5.6.25	ertc_wakeup_counter_get function	150
5.6.26	ertc_wakeup_enable function	151
5.6.27	ertc_smooth_calibration_config function	151
5.6.28	ertc_cal_output_select function	152
5.6.29	ertc_cal_output_enable function	152
5.6.30	ertc_time_adjust function	153
5.6.31	ertc_daylight_set function	153
5.6.32	ertc_daylight_bpr_get function.....	154
5.6.33	ertc_refer_clock_detect_enable function	154
5.6.34	ertc_direct_read_enable function.....	155
5.6.35	ertc_output_set function.....	155
5.6.36	ertc_timestamp_valid_edge_set function	156
5.6.37	ertc_timestamp_enable function	156
5.6.38	ertc_timestamp_get function	157
5.6.39	ertc_timestamp_sub_second_get function	158
5.6.40	ertc_tamper_pull_up_enable function.....	158
5.6.41	ertc_tamper_precharge_set function	159
5.6.42	ertc_tamper_filter_set function.....	159
5.6.43	ertc_tamper_detect_freq_set function	160
5.6.44	ertc_tamper_valid_edge_set function.....	161
5.6.45	ertc_tamper_timestamp_enable function.....	161
5.6.46	ertc_tamper_enable function.....	162
5.6.47	ertc_interrupt_enable function	162
5.6.48	ertc_interrupt_get function	163
5.6.49	ertc_flag_get function	163
5.6.50	ertc_interrupt_flag_get function	164
5.6.51	ertc_flag_clear function	165
5.6.52	ertc_bpr_data_write function.....	166
5.6.53	ertc_bpr_data_read function	166

5.7	External interrupt/event controller (EXINT)	167
5.7.1	exint_reset function	168
5.7.2	exint_default_para_init function	168
5.7.3	exint_init function.....	169
5.7.4	exint_flag_clear function	170
5.7.5	exint_flag_get function	170
5.7.6	exint_interrupt_flag_get function.....	171
5.7.7	exint_software_interrupt_event_generate function.....	171
5.7.8	exint_interrupt_enable function.....	172
5.7.9	exint_event_enable function	172
5.8	Flash memory controller (FLASH)	173
5.8.1	flash_flag_get function	175
5.8.2	flash_flag_clear function	176
5.8.3	flash_operation_status_get function	176
5.8.4	flash_operation_wait_for function	177
5.8.5	flash_unlock function.....	177
5.8.6	flash_lock function.....	178
5.8.7	flash_sector_erase function	178
5.8.8	flash_internal_all_erase function	179
5.8.9	flash_user_system_data_erase function	179
5.8.10	flash_word_program function.....	180
5.8.11	flash_halfword_program function.....	181
5.8.12	flash_byte_program function.....	182
5.8.13	flash_user_system_data_program function.....	183
5.8.14	flash_epp_set function	184
5.8.15	flash_epp_status_get function	184
5.8.16	flash_fap_enable function	185
5.8.17	flash_fap_status_get function	185
5.8.18	flash_fap_high_level_enable	186
5.8.19	flash_fap_high_level_status_get.....	186
5.8.20	flash_ssb_set function.....	187
5.8.21	flash_ssb_status_get function.....	188
5.8.22	flash_interrupt_enable function.....	188
5.8.23	flash_slib_enable function.....	189
5.8.24	flash_slib_disable function	190

5.8.25	flash_slib_state_get function.....	190
5.8.26	flash_slib_start_sector_get function.....	191
5.8.27	flash_slib_inststart_sector_get function.....	191
5.8.28	flash_slib_end_sector_get function.....	192
5.8.29	flash_crc_calibrate function.....	192
5.8.30	flash_boot_memory_extension_mode_enable.....	193
5.8.31	flash_extension_memory_slib_enable.....	193
5.8.32	flash_extension_memory_slib_state_get.....	194
5.8.33	flash_em_slib_inststart_sector_get.....	194
5.9	General-purpose I/Os and multiplexed I/Os (GPIO/IOMUX).....	195
5.9.1	gpio_reset function.....	196
5.9.2	gpio_init function.....	196
5.9.3	gpio_default_para_init function.....	198
5.9.4	gpio_input_data_bit_read function.....	198
5.9.5	gpio_input_data_read function.....	199
5.9.6	gpio_output_data_bit_read function.....	199
5.9.7	gpio_output_data_read function.....	200
5.9.8	gpio_bits_set function.....	200
5.9.9	gpio_bits_reset function.....	201
5.9.10	gpio_bits_toggle function.....	201
5.9.11	gpio_bits_write function.....	202
5.9.12	gpio_port_write function.....	202
5.9.13	gpio_pin_wp_config function.....	203
5.9.14	gpio_pins_huge_driven_config function.....	203
5.9.15	gpio_pin_mux_config function.....	204
5.10	Hardware integer divider (HWDIV).....	206
5.10.1	hwdiv_dividend_set function.....	207
5.10.2	hwdiv_divisor_set function.....	207
5.10.3	hwdiv_quotient_set function.....	208
5.10.4	hwdiv_remainder_get function.....	208
5.10.5	hwdiv_interrupt_enale function.....	209
5.10.6	hwdiv_signed_operate_enablefunction.....	209
5.10.7	hwdiv_flag_get_enable function.....	210
5.10.8	hwdiv_interrupt_flag_get function.....	210
5.10.9	hwdiv_flag_clear function.....	211

5.11	I2C interfaces	212
5.11.1	i2c_reset function	214
5.11.2	i2c_init function	215
5.11.3	i2c_own_address1_set function.....	215
5.11.4	i2c_own_address2_set function.....	216
5.11.5	i2c_own_address2_enable function.....	217
5.11.6	i2c_smbus_enable function.....	217
5.11.7	i2c_enable function	218
5.11.8	i2c_clock_stretch_enable function	218
5.11.9	i2c_ack_enable function.....	219
5.11.10	i2c_addr10_mode_enable function.....	219
5.11.11	i2c_transfer_addr_set function.....	220
5.11.12	i2c_transfer_addr_get function	220
5.11.13	i2c_transfer_dir_set function	221
5.11.14	i2c_transfer_dir_get function.....	221
5.11.15	i2c_matched_addr_get function.....	222
5.11.16	i2c_auto_stop_enable function	222
5.11.17	i2c_reload_enable function	223
5.11.18	i2c_cnt_set function.....	223
5.11.19	i2c_addr10_header_enable function.....	224
5.11.20	i2c_general_call_enable function.....	224
5.11.21	i2c_smbus_alert_set function	225
5.11.22	i2c_slave_data_ctrl_enable function.....	225
5.11.23	i2c_pec_calculate_enable function	226
5.11.24	i2c_pec_transmit_enable function	226
5.11.25	i2c_pec_value_get function.....	227
5.11.26	i2c_timeout_set function	227
5.11.27	i2c_timeout_detcet_set function	228
5.11.28	i2c_timeout_enable function	228
5.11.29	i2c_ext_timeout_set function.....	229
5.11.30	i2c_ext_timeout_enable function	229
5.11.31	i2c_interrupt_enable function.....	230
5.11.32	i2c_interrupt_get function.....	231
5.11.33	i2c_dma_enable function	232
5.11.34	i2c_transmit_set function	232

5.11.35 i2c_start_generate function	233
5.11.36 i2c_stop_generate function	234
5.11.37 i2c_data_send function	234
5.11.38 i2c_data_receive function	235
5.11.39 i2c_flag_get function	235
5.11.40 i2c_interrupt_flag_get function	236
5.11.41 i2c_flag_clear function	237
5.11.42 i2c_wakeup_enable function	238
5.11.43 i2c_analog_filter_enable function	238
5.11.44 i2c_config function	239
5.11.45 i2c_lowlevel_init function	241
5.11.46 i2c_wait_end function	242
5.11.47 i2c_wait_flag function	243
5.11.48 i2c_master_transmit function	244
5.11.49 i2c_master_receive function	245
5.11.50 i2c_slave_transmit function	246
5.11.51 i2c_slave_receive function	246
5.11.52 i2c_master_transmit_int function	247
5.11.53 i2c_master_receive_int function	248
5.11.54 i2c_slave_transmit_int function	249
5.11.55 i2c_slave_receive_int function	249
5.11.56 i2c_master_transmit_dma function	250
5.11.57 i2c_master_receive_dma function	251
5.11.58 i2c_slave_transmit_dma function	252
5.11.59 i2c_slave_receive_dma function	253
5.11.60 i2c_smbus_master_transmit function	254
5.11.61 i2c_smbus_master_receive function	255
5.11.62 i2c_smbus_slave_transmit function	256
5.11.63 i2c_smbus_slave_receive function	257
5.11.64 i2c_memory_write function	258
5.11.65 i2c_memory_write_int function	259
5.11.66 i2c_memory_write_dma function	260
5.11.67 i2c_memory_read function	261
5.11.68 i2c_memory_read_int function	262
5.11.69 i2c_memory_read_dma function	263

5.11.70	i2c_evt_irq_handler function	264
5.11.71	i2c_err_irq_handler function.....	264
5.11.72	i2c_dma_tx_irq_handler function	265
5.11.73	i2c_dma_rx_irq_handler function	265
5.12	Nested vectored interrupt controller (NVIC).....	266
5.12.1	nvic_system_reset function.....	267
5.12.2	nvic_irq_enable function	267
5.12.3	nvic_irq_disable function.....	268
5.12.4	nvic_vector_table_set function.....	268
5.12.5	nvic_lowpower_mode_config function	269
5.13	System configuration controller (SCFG)	270
5.13.1	scfg_reset function	271
5.13.2	scfg_infrared_config function	271
5.13.3	scfg_mem_map_get function.....	272
5.13.4	scfg_pa11pa12_pin_remap function.....	272
5.13.5	scfg_exint_line_config function	273
5.13.6	scfg_sram_operr_lock_enable function	274
5.13.7	scfg_pins_ultra_driven_enable function	274
5.13.8	scfg_pvm_lock_enable function.....	275
5.13.9	scfg_sram_operr_status_get function.....	275
5.13.10	scfg_lockup_enable function.....	276
5.14	Serial peripheral interface (SPI)/ I ² S	277
5.14.1	spi_i2s_reset function	278
5.14.2	spi_default_para_init function	278
5.14.3	spi_init function.....	279
5.14.4	spi_ti_mode_enable function	281
5.14.5	spi_crc_next_transmit function	281
5.14.6	spi_crc_polynomial_set function	282
5.14.7	spi_crc_polynomial_get function.....	282
5.14.8	spi_crc_enable function	283
5.14.9	spi_crc_value_get function.....	283
5.14.10	spi_hardware_cs_output_enable function	284
5.14.11	spi_software_cs_internal_level_set function	284
5.14.12	spi_frame_bit_num_set function	285
5.14.13	spi_half_duplex_direction_set function.....	285

5.14.14	spi_enable function	286
5.14.15	i2s_default_para_init function	286
5.14.16	i2s_init function.....	287
5.14.17	i2s_enable function	288
5.14.18	spi_i2s_interrupt_enable function	289
5.14.19	spi_i2s_dma_transmitter_enable function	290
5.14.20	spi_i2s_dma_receiver_enable function.....	290
5.14.21	spi_i2s_data_transmit function	291
5.14.22	spi_i2s_data_receive function.....	291
5.14.23	spi_i2s_flag_get function.....	292
5.14.24	spi_i2s_interrupt_flag_get function	293
5.14.25	spi_i2s_flag_clear function	294
5.15	SysTick.....	295
5.15.1	systick_clock_source_config function	295
5.15.2	SysTick_Config function.....	296
5.16	TMR	297
5.16.1	tmr_reset function.....	299
5.16.2	tmr_counter_enable function.....	299
5.16.3	tmr_output_default_para_init function.....	300
5.16.4	tmr_input_default_para_init function.....	300
5.16.5	tmr_brkdt_default_para_init function.....	301
5.16.6	tmr_base_init function	302
5.16.7	tmr_clock_source_div_set function.....	302
5.16.8	tmr_cnt_dir_set function.....	303
5.16.9	tmr_repetition_counter_set function.....	303
5.16.10	tmr_counter_value_set function.....	304
5.16.11	tmr_counter_value_get function.....	304
5.16.12	tmr_div_value_set function	305
5.16.13	tmr_div_value_get function	305
5.16.14	tmr_output_channel_config function	306
5.16.15	tmr_output_channel_mode_select function	308
5.16.16	tmr_period_value_set function.....	309
5.16.17	tmr_period_value_get function.....	309
5.16.18	tmr_channel_value_set function	310
5.16.19	tmr_channel_value_get function	311

5.16.20 tmr_period_buffer_enable function	311
5.16.21 tmr_output_channel_buffer_enable function	312
5.16.22 tmr_output_channel_immediately_set function	313
5.16.23 tmr_output_channel_switch_set function.....	314
5.16.24 tmr_one_cycle_mode_enable function	314
5.16.25 tmr_overflow_request_source_set function	315
5.16.26 tmr_overflow_event_disable function.....	315
5.16.27 tmr_input_channel_init function	316
5.16.28 tmr_channel_enable function.....	317
5.16.29 tmr_input_channel_filter_set function	318
5.16.30 tmr_pwm_input_config function	318
5.16.31 tmr_channel1_input_select function	319
5.16.32 tmr_input_channel_divider_set function	320
5.16.33 tmr_primary_mode_select function.....	321
5.16.34 tmr_sub_mode_select function	322
5.16.35 tmr_channel_dma_select function	323
5.16.36 tmr_hall_select function	323
5.16.37 tmr_channel_buffer_enable function.....	324
5.16.38 tmr_trgout2_enable function	324
5.16.39 tmr_trigger_input_select function.....	325
5.16.40 tmr_sub_sync_mode_set function	326
5.16.41 tmr_dma_request_enable function	326
5.16.42 tmr_interrupt_enable function	327
5.16.43 tmr_interrupt_flag_get function	328
5.16.44 tmr_flag_get function.....	329
5.16.45 tmr_flag_clear function.....	330
5.16.46 tmr_event_sw_trigger function.....	330
5.16.47 tmr_output_enable function.....	331
5.16.48 tmr_internal_clock_set function	331
5.16.49 tmr_output_channel_polarity_set function	332
5.16.50 tmr_external_clock_config function.....	333
5.16.51 tmr_external_clock_mode1_config function	334
5.16.52 tmr_external_clock_mode2_config function	335
5.16.53 tmr_encoder_mode_config function.....	336
5.16.54 tmr_force_output_set function	337

5.16.55	tmr_dma_control_config function.....	338
5.16.56	tmr_brkdt_config function.....	339
5.16.57	tmr_brk_filter_value_set function.....	341
5.16.58	tmr_iremap_config function.....	341
5.17	Watchdog timer (WDT).....	342
5.17.1	wdt_enable function.....	343
5.17.2	wdt_counter_reload function.....	343
5.17.3	wdt_reload_value_set function.....	344
5.17.4	wdt_divider_set function.....	344
5.17.5	wdt_register_write_enable function.....	345
5.17.6	wdt_flag_get function.....	345
5.17.7	wdt_window_counter_set function.....	346
5.18	Universal synchronous/asynchronous receiver/transmitter (USART).....	347
5.18.1	usart_reset function.....	348
5.18.2	usart_init function.....	349
5.18.3	usart_parity_selection_config function.....	350
5.18.4	usart_enable function.....	351
5.18.5	usart_transmitter_enable function.....	351
5.18.6	usart_receiver_enable function.....	352
5.18.7	usart_clock_config function.....	352
5.18.8	usart_clock_enable function.....	353
5.18.9	usart_interrupt_enable function.....	354
5.18.10	usart_dma_transmitter_enable function.....	355
5.18.11	usart_dma_receiver_enable function.....	356
5.18.12	usart_wakeup_id_set function.....	356
5.18.13	usart_wakeup_mode_set function.....	357
5.18.14	usart_receiver_mute_enable function.....	357
5.18.15	usart_break_bit_num_set function.....	358
5.18.16	usart_lin_mode_enable function.....	358
5.18.17	usart_data_transmit function.....	359
5.18.18	usart_data_receive function.....	359
5.18.19	usart_break_send function.....	360
5.18.20	usart_smartcard_guard_time_set function.....	360
5.18.21	usart_irda_smartcard_division_set function.....	361
5.18.22	usart_smartcard_mode_enable function.....	361

5.18.23	usart_smartcard_nack_set function.....	362
5.18.24	usart_single_line_halfduplex_select function	362
5.18.25	usart_irda_mode_enable function.....	363
5.18.26	usart_irda_low_power_enable function	363
5.18.27	usart_hardware_flow_control_set function	364
5.18.28	usart_flag_get function.....	364
5.18.29	usart_interrupt_flag_get function	365
5.18.30	usart_flag_clear function	366
5.18.31	usart_rs485_delay_time_config function	367
5.18.32	usart_transmit_receive_pin_swap function.....	367
5.18.33	usart_id_bit_num_set function	368
5.18.34	usart_de_polarity_set function.....	368
5.18.35	usart_rs485_mode_enable function.....	369
5.18.36	usart_low_power_wakeup_set function.....	370
5.18.37	usart_deep_sleep_mode_enable function.....	371
5.18.38	usart_msb_transmit_first_enable function	372
5.18.39	usart_dt_polarity_reverse function.....	373
5.18.40	usart_transmit_pin_polarity_reverse function	373
5.18.41	usart_receive_pin_polarity_reverse function	374
5.18.42	usart_receiver_timeout_detection_enable function	374
5.18.43	usart_receiver_timeout_value_set function	375
5.19	Window watchdog timer (WWDT).....	376
5.19.1	wwdt_reset function.....	377
5.19.2	wwdt_divider_set function	377
5.19.3	wwdt_enable function.....	378
5.19.4	wwdt_interrupt_enable function	378
5.19.5	wwdt_counter_set function.....	378
5.19.6	wwdt_window_counter_set function	379
5.19.7	wwdt_flag_get function.....	379
5.19.8	wwdt_interrupt_flag_get function	380
5.19.9	wwdt_flag_clear function.....	380
5.20	Power controller (PWC).....	381
5.20.1	pwc_reset function	382
5.20.2	pwc_battery_powered_domain_access function	382
5.20.3	pwc_pvm_level_select function	383

5.20.4	pwc_power_voltage_monitor_enable function.....	383
5.20.5	pwc_wakeup_pin_enable function	384
5.20.6	pwc_flag_clear function.....	385
5.20.7	pwc_flag_get function	385
5.20.8	pwc_sleep_mode_enter function	386
5.20.9	pwc_deep_sleep_mode_enter function	387
5.20.10	pwc_voltage_regulate_set function.....	387
5.20.11	pwc_standby_mode_enter function	388
5.20.12	pwc_ldo_output_voltage_set function.....	388
5.21	Analog-to-digital converter (ADC).....	389
5.21.1	adc_reset function	391
5.21.2	adc_enable function	391
5.21.3	adc_base_default_para_init function	392
5.21.4	adc_base_config function	392
5.21.5	adc_resolution_set function	394
5.21.6	adc_dma_mode_enable function.....	394
5.21.7	adc_interrupt_enable function.....	395
5.21.8	adc_calibration_init function.....	395
5.21.9	adc_calibration_init_status_get function	396
5.21.10	adc_calibration_start function	396
5.21.11	adc_calibration_status_get function.....	397
5.21.12	adc_voltage_monitor_enable function	397
5.21.13	adc_voltage_monitor_threshold_value_set function	398
5.21.14	adc_voltage_monitor_single_channel_select function	399
5.21.15	adc_ordinary_channel_set function	400
5.21.16	adc_preempt_channel_length_set function	401
5.21.17	adc_preempt_channel_set function	402
5.21.18	adc_ordinary_conversion_trigger_set function	403
5.21.19	adc_preempt_conversion_trigger_set function.....	404
5.21.20	adc_preempt_offset_value_set function	405
5.21.21	adc_ordinary_part_count_set function.....	406
5.21.22	adc_ordinary_part_mode_enable function	406
5.21.23	adc_preempt_part_mode_enable function	407
5.21.24	adc_preempt_auto_mode_enable function	407
5.21.25	adc_tempsensor_vintrv_enable function.....	408

5.21.26	adc_ordinary_software_trigger_enable function.....	408
5.21.27	adc_ordinary_software_trigger_status_get function.....	409
5.21.28	adc_preempt_software_trigger_enable function.....	409
5.21.29	adc_preempt_software_trigger_status_get function.....	410
5.21.30	adc_ordinary_conversion_data_get function.....	410
5.21.31	adc_preempt_conversion_data_get function.....	411
5.21.32	adc_flag_get function.....	412
5.21.33	adc_interrupt_flag_get function.....	413
5.21.34	adc_flag_clear function.....	413
5.21.35	adc_ordinary_oversample_enable function.....	414
5.21.36	adc_preempt_oversample_enable function.....	415
5.21.37	adc_oversample_ratio_shift_set function.....	415
5.21.38	adc_ordinary_oversample_trig_enable function.....	416
5.21.39	adc_ordinary_oversample_restart_set function.....	417
5.21.40	adc_precharge_enable function.....	418

6	Revision history.....	419
----------	------------------------------	------------

List of tables

Table 1. Summary of macro definitions	53
Table 2. List of abbreviations for peripherals	55
Table 3. Summary of BSP function library files	61
Table 4. Function format description for peripherals	62
Table 5. Summary of ADC registers	63
Table 6. Summary of CAN library functions.....	64
Table 7. can_reset function.....	65
Table 8. can_baudrate_default_para_init function	65
Table 9. can_baudrate_set function.....	66
Table 10. can_default_para_init function.....	67
Table 11. can_base_init function.....	67
Table 12. can_filter_default_para_init function	69
Table 13. can_filter_init function	69
Table 14. can_debug_transmission_prohibit function	71
Table 15. can_ttc_mode_enable function.....	71
Table 16. can_message_transmit function	72
Table 17. can_transmit_status_get function.....	74
Table 18. can_transmit_cancel function	75
Table 19. can_message_receive function	75
Table 20. can_receive_fifo_release function	77
Table 21. can_receive_message_pending_get function	78
Table 22. can_operating_mode_set function.....	78
Table 23. can_doze_mode_enter function	79
Table 24. can_doze_mode_exit function	80
Table 25. can_error_type_record_get function.....	80
Table 26. can_receive_error_counter_get function	81
Table 27. can_transmit_error_counter_get function.....	81
Table 28. can_interrupt_enable function	82
Table 29. can_flag_get function.....	83
Table 30. can_interrupt_flag_get function	84
Table 31. can_flag_clear function	85
Table 32. Summary of CRC registers	86
Table 33. Summary of CRC library functions	86
Table 34. crc_data_reset function.....	87

Table 35. crc_one_word_calculate function	87
Table 36. crc_block_calculate function	88
Table 37. crc_data_get function.....	88
Table 38. crc_common_data_set function.....	89
Table 39. crc_common_data_get function.....	89
Table 40. crc_init_data_set function	90
Table 41. crc_reverse_input_data_set function.....	90
Table 42. crc_reverse_output_data_set function.....	91
Table 43. crc_poly_value_set function.....	91
Table 44. crc_poly_value_get function	92
Table 45. crc_poly_size_set function.....	92
Table 46. crc_poly_size_get function.....	93
Table 47. Summary of CRM registers.....	94
Table 48. Summary of CRM library functions	95
Table 49. crm_reset function.....	96
Table 50. crm_lxt_bypass function	96
Table 51. crm_hxt_bypass function	97
Table 52. crm_flag_get function.....	97
Table 53. crm_interrupt_flag_get function	98
Table 54. crm_hxt_stable_wait function.....	99
Table 55. crm_hick_clock_trimming_set function.....	99
Table 56. crm_hick_clock_calibration_set function	100
Table 57. crm_periph_clock_enable function	100
Table 58. crm_periph_reset function	101
Table 59. crm_periph_sleep_mode_clock_enable	101
Table 60. crm_clock_source_enable function	102
Table 61. crm_flag_clear function.....	103
Table 62. crm_ertc_clock_select function.....	104
Table 63. crm_ertc_clock_enable function	105
Table 64. crm_ahb_div_set function	105
Table 65. crm_apb1_div_set function	106
Table 66. crm_apb2_div_set function.....	106
Table 67. crm_adc_clock_div_set	107
Table 68. crm_clock_failure_detection_enable function.....	107
Table 69. crm_battery_powered_domain_reset	108

Table 70. crm_pll_config function	108
Table 71. crm_pll_config2 function	109
Table 72. crm_sysclk_switch function.....	110
Table 73. crm_sysclk_switch_status_get function.....	110
Table 74. crm_clocks_freq_get function	111
Table 75. crm_clock_out_set function	112
Table 76. crm_interrupt_enable function	112
Table 77. crm_hick_sclk_frequency_select function	113
Table 78. crm_clkout_div_set.....	114
Table 79. crm_usart_clock_select function.....	114
Table 80. crm_usart_clock_get function	115
Table 81. crm_i2c_clock_select function	116
Table 82. crm_i2c_clock_get function.....	117
Table 83. Summary of DEBUG registers	118
Table 84. Summary of DEBUG library functions	118
Table 85. debug_device_id_get function	118
Table 86. debug_device_series_id_getfunction	119
Table 87. debug_periph_mode_set function	120
Table 88. Summary of DMA registers	121
Table 89. Summary of DMA library functions	122
Table 90. dma_default_para_init function.....	122
Table 91. dma_init_struct default values	122
Table 92. dma_init function	123
Table 93. dma_reset function.....	125
Table 94. dma_data_number_set function	125
Table 95. dma_data_number_get function	126
Table 96. dma_interrupt_enable function	126
Table 97. dma_channel_enable function.....	127
Table 98. dma_flexible_config function.....	127
Table 99. DMAMUX channel request source ID.....	128
Table 100. dma_flag_get function.....	129
Table 101. dma_interrupt_flag_get function	130
Table 102. dma_flag_clear function.....	131
Table 103. Summary of ERTC registers	132
Table 104. Summary of ERTC library functions	133

Table 105. ertc_num_to_bcd function.....	135
Table 106. ertc_bcd_to_num function.....	135
Table 107. ertc_write_protect_enable function.....	136
Table 108. ertc_write_protect_disable function	136
Table 109. ertc_wait_update function	137
Table 110. ertc_wait_flag function.....	137
Table 111. ertc_init_mode_enter function	138
Table 112. ertc_init_mode_exit function	138
Table 113. ertc_reset function.....	139
Table 114. ertc_divider_set function	139
Table 115. ertc_hour_mode_set function.....	140
Table 116. ertc_date_set function	140
Table 117. ertc_time_set function	141
Table 118. ertc_calendar_get function.....	142
Table 119. ertc_sub_second_get function	143
Table 120. ertc_alarm_mask_set function.....	143
Table 121. ertc_alarm_week_date_select function	144
Table 122. ertc_alarm_set function.....	145
Table 123. ertc_alarm_sub_second_set function.....	146
Table 124. ertc_alarm_enable function.....	147
Table 125. ertc_alarm_get function	147
Table 126. ertc_alarm_sub_second_get function.....	149
Table 127. ertc_wakeup_clock_set function.....	149
Table 128. ertc_wakeup_counter_set function	150
Table 129. ertc_wakeup_counter_get function.....	150
Table 130. ertc_wakeup_enable function	151
Table 131. ertc_smooth_calibration_config function	151
Table 132. ertc_cal_output_select function	152
Table 133. ertc_cal_output_enable function.....	152
Table 134. ertc_time_adjust function	153
Table 135. ertc_daylight_set function	153
Table 136. ertc_daylight_bpr_get function.....	154
Table 137. ertc_refer_clock_detect_enable function.....	154
Table 138. ertc_direct_read_enable function	155
Table 139. ertc_output_set function.....	155

Table 140. ertc_timestamp_valid_edge_set function	156
Table 141. ertc_timestamp_enable function.....	156
Table 142. ertc_timestamp_get function.....	157
Table 143. ertc_timestamp_sub_second_get function	158
Table 144. ertc_tamper_pull_up_enable function.....	158
Table 145. ertc_tamper_precharge_set function	159
Table 146. ertc_tamper_filter_set function	159
Table 147. ertc_tamper_detect_freq_set function	160
Table 148. ertc_tamper_valid_edge_set function.....	161
Table 149. ertc_tamper_timestamp_enable function	161
Table 150. ertc_tamper_enable function	162
Table 151. ertc_interrupt_enable function	162
Table 152. ertc_interrupt_get function	163
Table 153. ertc_flag_get function.....	163
Table 154. ertc_interrupt_flag_get function	164
Table 155. ertc_flag_clear function.....	165
Table 156. ertc_bpr_data_write function	166
Table 157. ertc_bpr_data_read function.....	166
Table 158. Summary of EXINT registers	167
Table 159. Summary of EXINT library functions.....	167
Table 160. exint_reset function	168
Table 161. exint_default_para_init function	168
Table 162. exint_init function	169
Table 163. exint_flag_clear function	170
Table 164. exint_flag_get function	170
Table 165. exint_interrupt_flag_get function.....	171
Table 166. exint_software_interrupt_event_generate function	171
Table 167. exint_interrupt_enable function.....	172
Table 168. exint_event_enable function	172
Table 169. Summary of FLASH registers	173
Table 170. Summary of FLASH library functions.....	174
Table 171. flash_flag_get function	175
Table 172. flash_flag_clear function	176
Table 173. flash_operation_status_get function.....	176
Table 174. flash_operation_wait_for function	177

Table 175. flash_unlock function	177
Table 176. flash_lock function.....	178
Table 177. flash_sector_erase function	178
Table 178. flash_internal_all_erase function	179
Table 179. flash_user_system_data_erase function	179
Table 180. flash_word_program function	180
Table 181. flash_halfword_program function.....	181
Table 182. flash_byte_program function.....	182
Table 183. flash_user_system_data_program function.....	183
Table 184. flash_epp_set function	184
Table 185. flash_epp_status_get function	184
Table 186. flash_fap_enable function	185
Table 187. flash_fap_status_get function	185
Table 188. flash_fap_high_level_enable function.....	186
Table 189. flash_fap_high_level_status_get function.....	186
Table 190. flash_ssb_set function	187
Table 191. flash_ssb_status_get function	188
Table 192. flash_interrupt_enable function.....	188
Table 193. flash_slib_enable function.....	189
Table 194. flash_slib_disable function	190
Table 195. flash_slib_state_get function	190
Table 196. flash_slib_start_sector_get function	191
Table 197. flash_slib_inststart_sector_get function.....	191
Table 198. flash_slib_end_sector_get function	192
Table 199. flash_crc_calibrate function	192
Table 200. flash_boot_memory_extension_mode_enable.....	193
Table 201. flash_extension_memory_slib_enable	193
Table 202. flash_extension_memory_slib_state_get	194
Table 203. flash_em_slib_inststart_sector_get	194
Table 204. Summary of GPIO registers.....	195
Table 205. GPIO and IOMUX library functions.....	195
Table 206. gpio_reset function.....	196
Table 207. gpio_init function	196
Table 208. gpio_default_para_init function	198
Table 209. gpio_init_struct default values	198

Table 210. gpio_input_data_bit_read function.....	198
Table 211. gpio_input_data_read function.....	199
Table 212. gpio_output_data_bit_read function	199
Table 213. gpio_output_data_read function	200
Table 214. gpio_bits_set function	200
Table 215. gpio_bits_reset function	201
Table 216. gpio_bits_toggle function	201
Table 217. gpio_bits_write function	202
Table 218. gpio_port_write function.....	202
Table 219. gpio_pin_wp_config function	203
Table 220. gpio_pins_huge_driven_config function	203
Table 221. gpio_pin_mux_config function	204
Table 222. Summary of HWDIV registers.....	206
Table 223. Summary of HWDIV library functions	206
Table 224. hwdiv_dividend_set function.....	207
Table 225. hwdiv_divisor_set function.....	207
Table 226. hwdiv_quotient_get function	208
Table 227. hwdiv_quotient_get function	208
Table 228. hwdiv_interrupt_enale function	209
Table 229. hwdiv_signed_operate_enable function	209
Table 230. hwdiv_flag_get function	210
Table 231. hwdiv_interrupt_flag_get function.....	210
Table 232. hwdiv_flag_clear function.....	211
Table 233. Summary of I2C register	212
Table 234. Summary of I2C library functions.....	212
Table 235. I2C application-layer library functions.....	213
Table 236. i2c_reset function.....	214
Table 237. i2c_init function	215
Table 238. i2c_own_address1_set function	215
Table 239. i2c_own_address2_set function	216
Table 240. i2c_own_address2_enable function	217
Table 241. i2c_smbus_enable function	217
Table 242. i2c_enable function	218
Table 243. i2c_clock_stretch_enable function.....	218
Table 244. i2c_ack_enable function	219

Table 245. i2c_addr10_mode_enable function.....	219
Table 246. i2c_transfer_addr_set function	220
Table 247. i2c_transfer_addr_get function	220
Table 248. i2c_transfer_dir_set function.....	221
Table 249. i2c_transfer_dir_get function.....	221
Table 250. i2c_matched_addr_get function.....	222
Table 251. i2c_auto_stop_enable function	222
Table 252. i2c_reload_enable function	223
Table 253. i2c_cnt_set function	223
Table 254. i2c_addr10_header_enable function	224
Table 255. i2c_general_call_enable function	224
Table 256. i2c_smbus_alert_set function	225
Table 257. i2c_start_generate function.....	225
Table 258. i2c_pec_calculate_enable.....	226
Table 259. i2c_pec_transmit_enable function	226
Table 260. i2c_pec_value_get function	227
Table 261. i2c_timeout_set function	227
Table 262. i2c_timeout_detcet_set function	228
Table 263. i2c_timeout_enable function	228
Table 264. i2c_ext_timeout_set function	229
Table 265. i2c_ext_timeout_enable function	229
Table 266. i2c_interrupt_enable function.....	230
Table 267. i2c_interrupt_get function.....	231
Table 268. i2c_dma_enable function	232
Table 269. i2c_transmit_set function	232
Table 270. i2c_slave_transmit function.....	233
Table 271. i2c_stop_generate function.....	234
Table 272. i2c_data_send function	234
Table 273. i2c_data_receive function	235
Table 274. i2c_flag_get function	235
Table 275. i2c_interrupt_flag_get function.....	236
Table 276. i2c_flag_clear function	237
Table 277. i2c_wakeup_enable function	238
Table 278. i2c_analog_filter_enable function	238
Table 279. i2c_config function	239

Table 280. i2c_lowlevel_init function	241
Table 281. i2c_wait_end function	242
Table 282. i2c_wait_flag function.....	243
Table 283. i2c_master_transmit function	244
Table 284. i2c_master_receivefunction	245
Table 285. i2c_slave_transmit function.....	246
Table 286. i2c_slave_receive function.....	246
Table 287. i2c_master_transmit_int function	247
Table 288. i2c_master_receive_int function	248
Table 289. i2c_master_receive_int function	249
Table 290. i2c_master_receive_int function	249
Table 291. i2c_master_transmit_dma function.....	250
Table 292. i2c_master_receive_dma function	251
Table 293. i2c_slave_transmit_dma function	252
Table 294. i2c_slave_receive_dma function.....	253
Table 295. i2c_smbus_master_transmit function	254
Table 296. i2c_smbus_master_receive function	255
Table 297. i2c_smbus_slave_transmit function.....	256
Table 298. i2c_smbus_slave_receive function.....	257
Table 299. i2c_memory_write function	258
Table 300. i2c_memory_write_int function	259
Table 301. i2c_memory_write_dma function	260
Table 302. i2c_memory_write_dma function	261
Table 303. i2c_memory_write_dma function	262
Table 304. i2c_memory_write_dma function	263
Table 305. i2c_evt_irq_handler function	264
Table 306. i2c_err_irq_handler function	264
Table 307. i2c_dma_tx_irq_handler function.....	265
Table 308. i2c_dma_rx_irq_handler function.....	265
Table 309. Summary of NVIC registers	266
Table 310. Summary of NVIC library functions.....	266
Table 311. nvic_system_reset function.....	267
Table 312. nvic_irq_enable function	267
Table 313. nvic_irq_disable function.....	268
Table 314. nvic_vector_table_set function	268

Table 315. nvic_lowpower_mode_config function	269
Table 316. Summary of SCFG registers.....	270
Table 317. Summary of SCFG library functions	270
Table 318. scfg_reset function	271
Table 319. scfg_infrared_config function	271
Table 320. scfg_mem_map_get function.....	272
Table 321. scfg_pa11pa12_pin_remap	272
Table 322. scfg_exint_line_config	273
Table 323. scfg_sram_operr_lock_enable function.....	274
Table 324. scfg_pins_ultra_driven_enable function	274
Table 325. scfg_adc_dma_channel_remap function.....	275
Table 326. scfg_sram_operr_status_get function.....	275
Table 327. scfg_usart1_tx_dma_channel_remap function.....	276
Table 328. Summary of SPI registers	277
Table 329. Summary of SPI library functions	277
Table 330. spi_i2s_reset function	278
Table 331. spi_default_para_init function	278
Table 332. spi_init function	279
Table 333. spi_ti_mode_enable function	281
Table 334. spi_crc_next_transmit function	281
Table 335. spi_crc_polynomial_set function.....	282
Table 336. spi_crc_polynomial_get function.....	282
Table 337. spi_crc_enable function	283
Table 338. spi_crc_value_get function	283
Table 339. spi_hardware_cs_output_enable function	284
Table 340. spi_software_cs_internal_level_set function	284
Table 341. spi_frame_bit_num_set function.....	285
Table 342. spi_half_duplex_direction_set function.....	285
Table 343. spi_enable function	286
Table 344. i2s_default_para_init function	286
Table 345. i2s_init function	287
Table 346. i2s_enable function	288
Table 347. spi_i2s_interrupt_enable function.....	289
Table 348. spi_i2s_dma_transmitter_enable function	290
Table 349. spi_i2s_dma_receiver_enable function	290

Table 350. spi_i2s_data_transmit function	291
Table 351. spi_i2s_data_receive function	291
Table 352. spi_i2s_flag_get function	292
Table 353. spi_i2s_interrupt_flag_get function	293
Table 354. spi_i2s_flag_clear function.....	294
Table 355. Summary of SysTick registers	295
Table 356. Summary of SysTick library functions.....	295
Table 357. systick_clock_source_config function.....	295
Table 358. SysTick_Config function.....	296
Table 359. Summary of TMR registers	297
Table 360. Summary of TMR library functions	298
Table 361. tmr_reset function	299
Table 362. tmr_counter_enable function	299
Table 363. tmr_output_default_para_init function	300
Table 364. tmr_output_struct default values.....	300
Table 365. tmr_input_default_para_init function.....	300
Table 366. tmr_input_struct default values	301
Table 367. tmr_brkdt_default_para_init function	301
Table 368. tmr_brkdt_struct default values.....	301
Table 369. tmr_base_init function.....	302
Table 370. tmr_clock_source_div_set function.....	302
Table 371. tmr_cnt_dir_set function.....	303
Table 372. tmr_repetition_counter_set function	303
Table 373. tmr_counter_value_set function.....	304
Table 374. tmr_counter_value_get function	304
Table 375. tmr_div_value_set function	305
Table 376. tmr_div_value_get function	305
Table 377. tmr_output_channel_config function.....	306
Table 378. tmr_output_channel_mode_select function.....	308
Table 379. tmr_period_value_set function.....	309
Table 380. tmr_period_value_get function	309
Table 381. tmr_channel_value_set function	310
Table 382. tmr_channel_value_get function.....	311
Table 383. tmr_period_buffer_enable function	311
Table 384. tmr_output_channel_buffer_enable function	312

Table 385. tmr_output_channel_immediately_set function	313
Table 386. tmr_output_channel_switch_set function	314
Table 387. tmr_one_cycle_mode_enable function.....	314
Table 388. tmr_overflow_request_source_set function.....	315
Table 389. tmr_overflow_event_disable function	315
Table 390. tmr_input_channel_init function	316
Table 391. tmr_channel_enable function.....	317
Table 392. tmr_input_channel_filter_set function.....	318
Table 393. tmr_pwm_input_config function	318
Table 394. tmr_channel1_input_select function	319
Table 395. tmr_input_channel_divider_set function	320
Table 396. tmr_primary_mode_select function.....	321
Table 397. tmr_sub_mode_select function	322
Table 398. tmr_channel_dma_select function	323
Table 399. tmr_hall_select function	323
Table 400. tmr_channel_buffer_enable function	324
Table 401. tmr_trgout2_enable function	324
Table 402. tmr_trigger_input_select function.....	325
Table 403. tmr_sub_sync_mode_set function	326
Table 404. tmr_dma_request_enable function	326
Table 405. tmr_interrupt_enable function	327
Table 406. tmr_interrupt_flag_get function	328
Table 407. tmr_flag_get function	329
Table 408. tmr_flag_clear function.....	330
Table 409. tmr_event_sw_trigger function.....	330
Table 410. tmr_output_enable function	331
Table 411. tmr_internal_clock_set function.....	331
Table 412. tmr_output_channel_polarity_set function	332
Table 413. tmr_external_clock_config function	333
Table 414. tmr_external_clock_mode1_config function	334
Table 415. tmr_external_clock_mode2_config function	335
Table 416. tmr_encoder_mode_config function	336
Table 417. tmr_force_output_set function	337
Table 418. tmr_dma_control_config function.....	338
Table 419. tmr_brkdt_config function.....	339

Table 420. tmr_brk_filter_value_set function	341
Table 421. tmr_iremap_config function.....	341
Table 422. Summary of WDT registers.....	342
Table 423. Summary of WDT library functions	342
Table 424. wdt_enable function	343
Table 425. wdt_counter_reload function.....	343
Table 426. wdt_reload_value_set function	344
Table 427. wdt_divider_set function	344
Table 428. wdt_register_write_enable function	345
Table 429. wdt_flag_get function	345
Table 430. wdt_window_counter_set function.....	346
Table 431. Summary of USART registers.....	347
Table 432. Summary of USART library functions	347
Table 433. usart_reset function	348
Table 434. usart_init function.....	349
Table 435. usart_parity_selection_config function	350
Table 436. usart_enable function.....	351
Table 437. usart_transmitter_enable function	351
Table 438. usart_receiver_enable function.....	352
Table 439. usart_clock_config function.....	352
Table 440. usart_clock_enable function	353
Table 441. usart_interrupt_enable function	354
Table 442. usart_dma_transmitter_enable function	355
Table 443. usart_dma_receiver_enable function.....	356
Table 444. usart_wakeup_id_set function	356
Table 445. usart_wakeup_mode_set function	357
Table 446. usart_receiver_mute_enable function.....	357
Table 447. usart_break_bit_num_set function.....	358
Table 448. usart_lin_mode_enable function.....	358
Table 449. usart_data_transmit function.....	359
Table 450. usart_data_receive function.....	359
Table 451. usart_break_send function.....	360
Table 452. usart_smartcard_guard_time_set function	360
Table 453. usart_irda_smartcard_division_set function	361
Table 454. usart_smartcard_mode_enable function	361

Table 455. usart_smartcard_nack_set function.....	362
Table 456. usart_single_line_halfduplex_select function	362
Table 457. usart_irda_mode_enable function	363
Table 458. usart_irda_low_power_enable function	363
Table 459. usart_hardware_flow_control_set function	364
Table 460. usart_flag_get function.....	364
Table 461. usart_interrupt_flag_get function	365
Table 462. usart_flag_clear function.....	366
Table 463. usart_flag_clear function.....	367
Table 464. usart_transmit_receive_pin_swap function	367
Table 465. usart_transmit_receive_pin_swap function	368
Table 466. usart_de_polarity_set function.....	368
Table 467. usart_rs485_mode_enable function	369
Table 468. usart_low_power_wakeup_set function.....	370
Table 469. usart_deep_sleep_mode_enable function.....	371
Table 470. usart_msb_transmit_first_enable function.....	372
Table 471. usart_dt_polarity_reverse function.....	373
Table 472. usart_dt_polarity_reverse function.....	373
Table 473. usart_receive_pin_polarity_reverse function	374
Table 474. usart_receiver_timeout_detection_enable function.....	374
Table 475. usart_receiver_timeout_value_set function	375
Table 476. Summary of WWDT registers	376
Table 477. Summary of WWDT library functions.....	376
Table 478. wwdt_reset function	377
Table 479. wwdt_divider_set function.....	377
Table 480. wwdt_enable function	378
Table 481. wwdt_interrupt_enable function	378
Table 482. wwdt_counter_set function	378
Table 483. wwdt_window_counter_set function	379
Table 484. wwdt_flag_get function	379
Table 485. wwdt_interrupt_flag_get function	380
Table 486. wwdt_flag_clear function.....	380
Table 487. Summary of PWC registers	381
Table 488. Summary of PWC library functions.....	381
Table 489. pwc_reset function	382

Table 490. pwc_battery_powered_domain_access function.....	382
Table 491. pwc_pvm_level_select function	383
Table 492. pwc_power_voltage_monitor_enable function	383
Table 493. pwc_wakeup_pin_enable function.....	384
Table 494. pwc_flag_clear function	385
Table 495. pwc_flag_get function	385
Table 496. pwc_sleep_mode_enter function	386
Table 497. pwc_deep_sleep_mode_enter function.....	387
Table 498. pwc_voltage_regulate_set function	387
Table 499. pwc_standby_mode_enter function.....	388
Table 500. pwc_lido_output_voltage_set function.....	388
Table 501. Summary of ADC registers	389
Table 502. Summary of ADC library functions.....	390
Table 503. adc_reset function.....	391
Table 504. adc_enable function.....	391
Table 505. adc_base_default_para_init function.....	392
Table 506. adc_base_config function	392
Table 507. adc_resolution_set function	394
Table 508. adc_dma_mode_enable function.....	394
Table 509. adc_interrupt_enable function	395
Table 510. adc_calibration_init function.....	395
Table 511. adc_calibration_init_status_get function.....	396
Table 512. adc_calibration_start function	396
Table 513. adc_calibration_status_get function	397
Table 514. adc_voltage_monitor_enable function.....	397
Table 515. adc_voltage_monitor_threshold_value_set function	398
Table 516. adc_voltage_monitor_single_channel_select function.....	399
Table 517. adc_ordinary_channel_set function.....	400
Table 518. adc_preempt_channel_length_set function.....	401
Table 519. adc_preempt_channel_set function.....	402
Table 520. adc_ordinary_conversion_trigger_set function.....	403
Table 521. adc_preempt_conversion_trigger_set function	404
Table 522. adc_preempt_offset_value_set function.....	405
Table 523. adc_ordinary_part_count_set function	406
Table 524. adc_ordinary_part_mode_enable function	406

Table 525. adc_preempt_part_mode_enable function	407
Table 526. adc_preempt_auto_mode_enable function	407
Table 527. adc_tempsensor_vintrv_enable.....	408
Table 528. adc_ordinary_software_trigger_enable function.....	408
Table 529. adc_ordinary_software_trigger_status_get function.....	409
Table 530. adc_preempt_software_trigger_enable function	409
Table 531. adc_preempt_software_trigger_status_get function	410
Table 532. adc_ordinary_conversion_data_get function.....	410
Table 533. adc_preempt_conversion_data_get function.....	411
Table 534. adc_flag_get function.....	412
Table 535. adc_interrupt_flag_get function	413
Table 536. adc_flag_clear function	413
Table 536. adc_ordinary_oversample_enable function.....	414
Table 537. adc_preempt_oversample_enable function.....	415
Table 538. adc_oversample_ratio_shift_set function	415
Table 539. adc_ordinary_oversample_trig_enable function.....	416
Table 540. adc_ordinary_oversample_restart_set function	417
Table 542. adc_precharge_enable function	418
Table 543. Document revision history.....	419

List of figures

Figure 1. Pack kit	36
Figure 2. IAR Pack installation window	36
Figure 3. IAR Pack installation window	37
Figure 4. View IAR Pack installation status	38
Figure 5. View Keil_v5 Pack installation status	39
Figure 6. Keil_v4 Pack installation.....	40
Figure 7. Keil_v4 Pack installation process	41
Figure 8. Keil_v4 Pack installation complete	41
Figure 9. View Keil_v4 Pack installation status	42
Figure 10. Segger pack installation window	43
Figure 11. Segger pack installation process.....	43
Figure 12. Open J-Flash	44
Figure 13. Create a new project using J-Flash.....	44
Figure 14. View Device information	44
Figure 15. Keil algorithm file settings.....	46
Figure 16. Keil algorithm file configuration	47
Figure 17. Select algorithm files using Keil	47
Figure 18. Add algorithm files using Keil	48
Figure 19. IAR project name.....	49
Figure 20. IAR algorithm file configuration	49
Figure 21. IAR Flash Loader overview	50
Figure 22. IAR Flash Loader configuration.....	50
Figure 23. IAR Flash Loader configuration success	51
Figure 24. Template content	52
Figure 25. Keil_v5 template project example	53
Figure 26. Peripheral enable macro definitions.....	54
Figure 27. BSP folder structure	59
Figure 28. BSP function library structure.....	59

1 Overview

In order to help users make efficient use of Artery MCU, we provide a complete set of BSP & Pack tools to speed up development. They include peripheral driver library, core-related documents and application cases as well as Pack documents supporting a variety of development environments such as Keil_v5, Keil_v4, IAR_v6, IAR_v7 and IAR_v8. The BSP and Pack are available on Artery official website.

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

2 How to install Pack

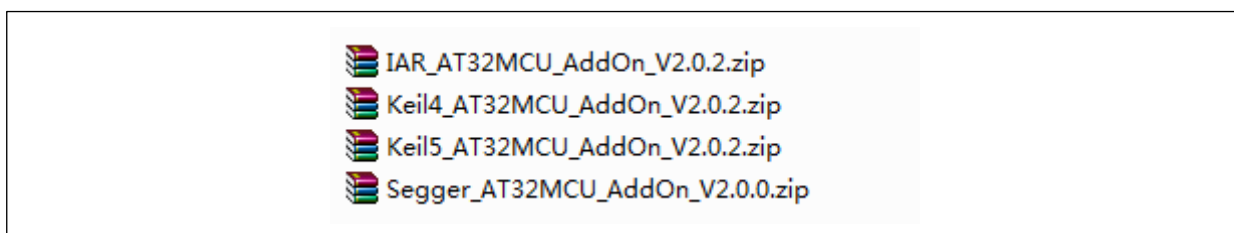
Artery Pack supports various development environment such as Keil_v5, Keil_v4, IAR_v6, IAR_v7 and IAR_v8.

Double click on the corresponding Pack to finish installation.

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

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

Figure 1. Pack kit

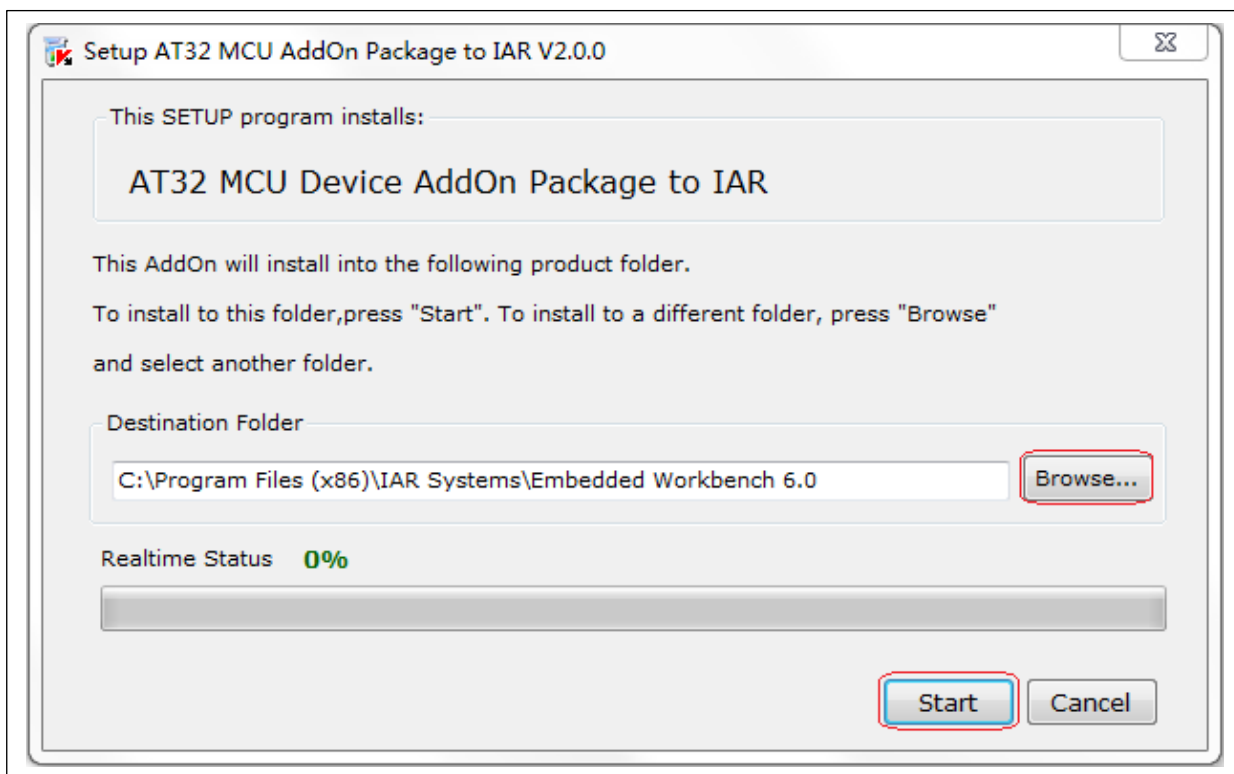


2.1 IAR Pack installation

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

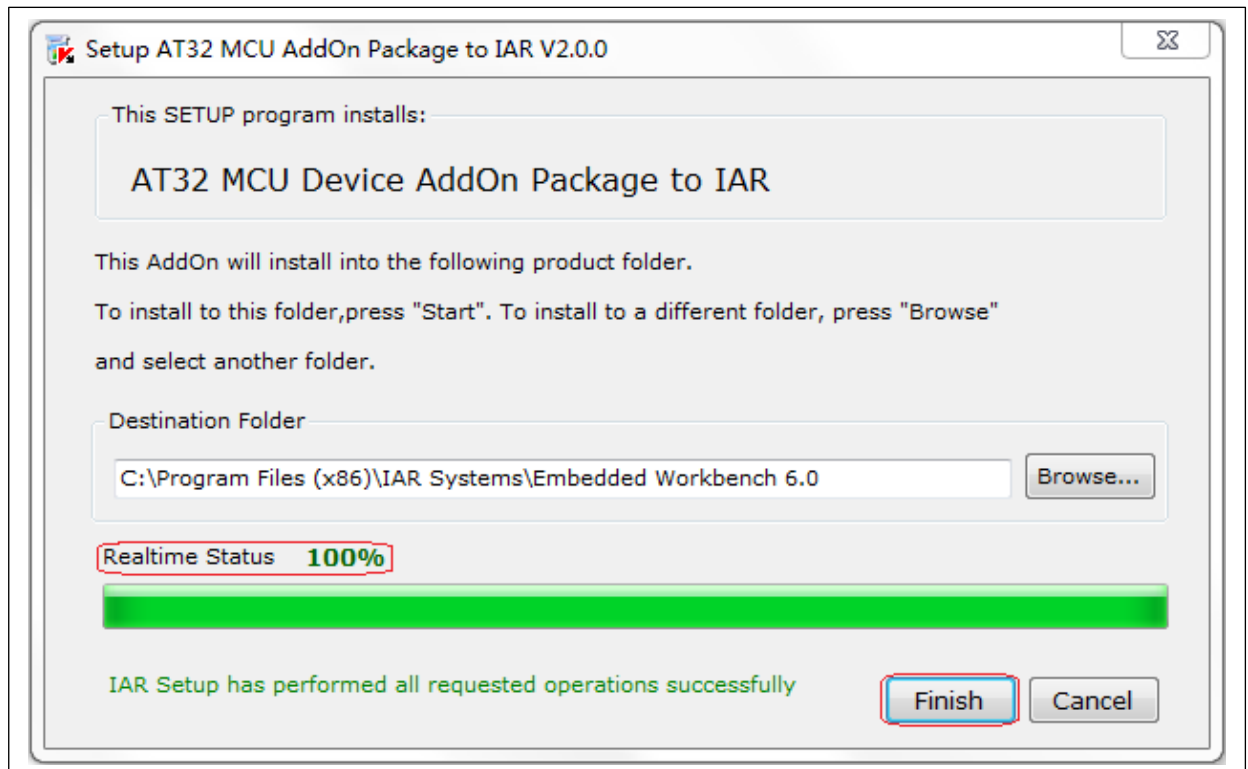
- ① Unzip *IAR_AT32MCU_AddOn.zip*;
- ② Double click on *IAR_AT32MCU_AddOn.exe*, and a dialog box pops up below (the specific version information is subject to the actual conditions).

Figure 2. IAR Pack installation window



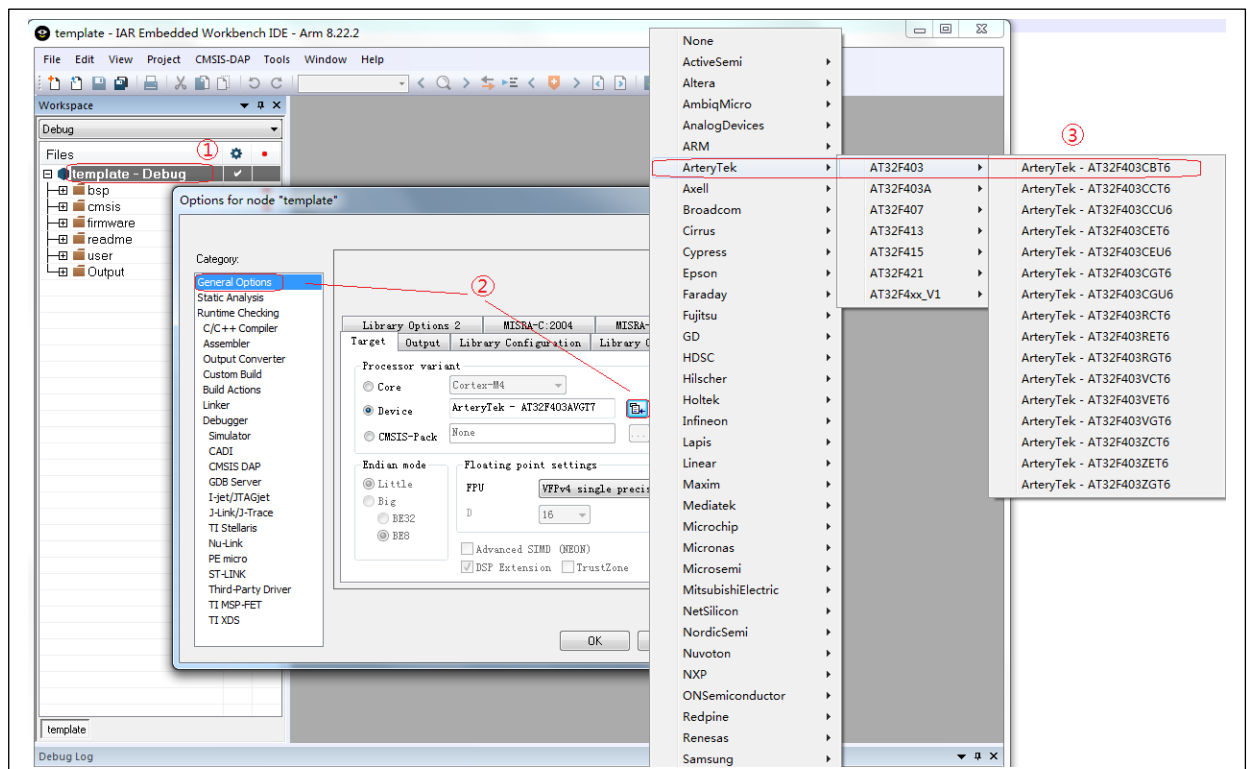
Note: If the installation path of IAR does not match the Destination Folder, click on "Browse" to select a correct path, then click on "Start", as shown below.

Figure 3. IAR Pack installation window



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

Figure 4. View IAR Pack installation status

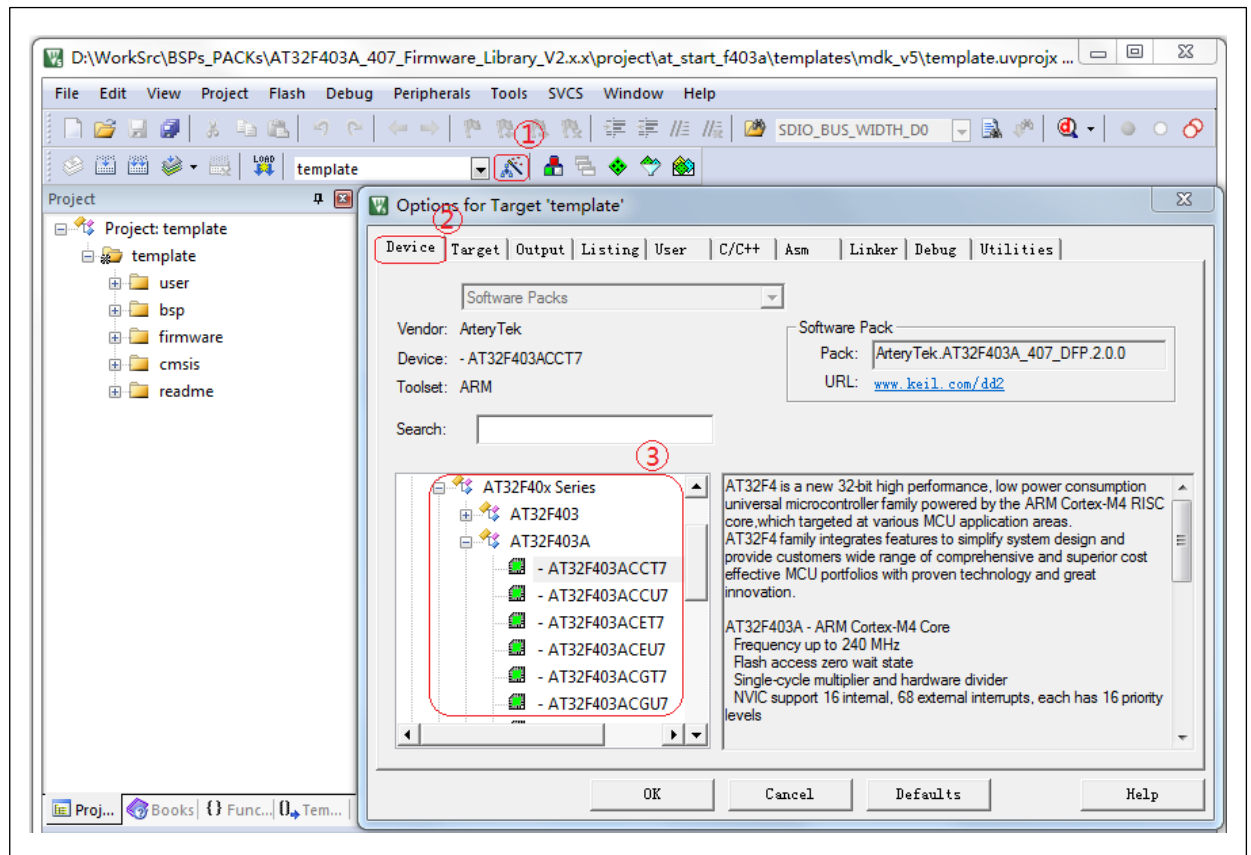


2.2 Keil_v5 Pack installation

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

- ① Unzip *Keil5_AT32MCU_AddOn.zip*. This zip file includes all Keil5 packs supported, all of which are standard Keil_v5 DFP installation files.
- ② Select the desired Pack, and double click on *ArteryTek.AT32xxxx_DFP.2.x.x.pack* to get one-stop installation.
- ⑤ To check whether the Keil_v5 Pack is installed successfully or not, follow the steps below:
 - Click on wand;
 - Select "Device";
 - View AT32 MCU-related information.

Figure 5. View Keil_v5 Pack installation status

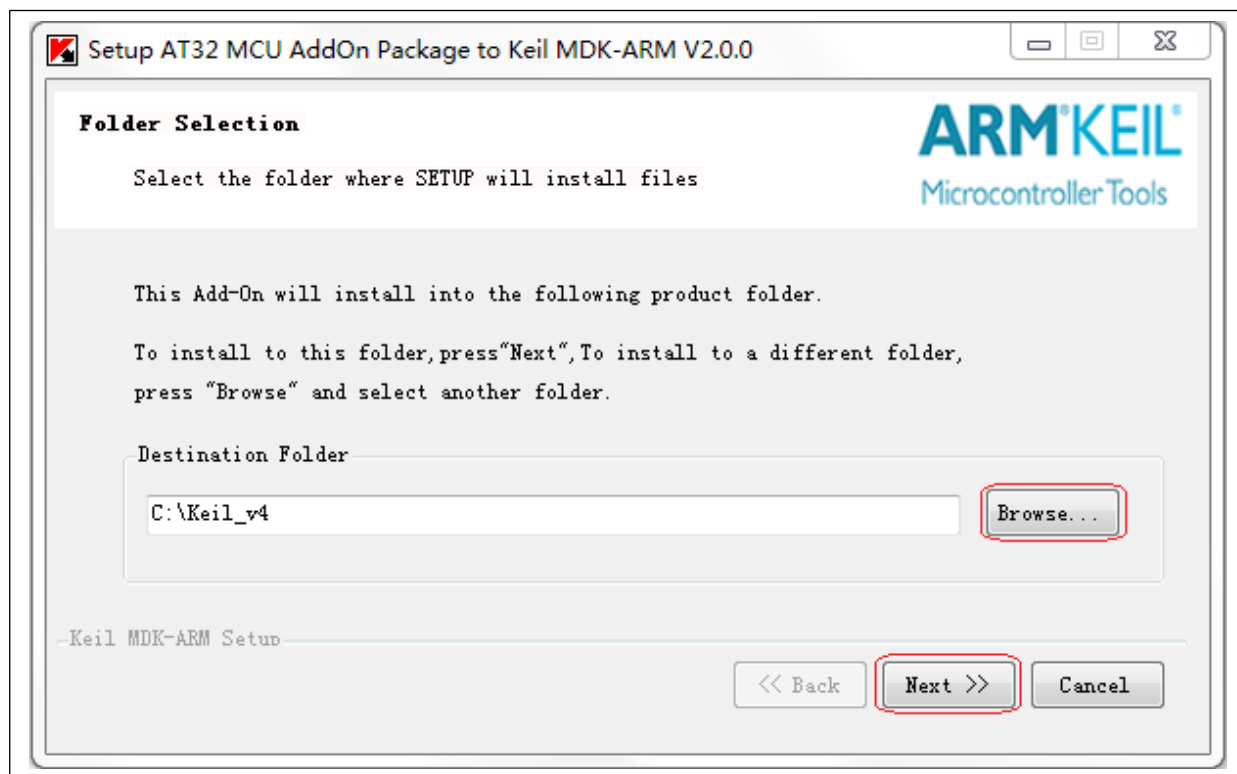


2.3 Keil_v4 Pack installation

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

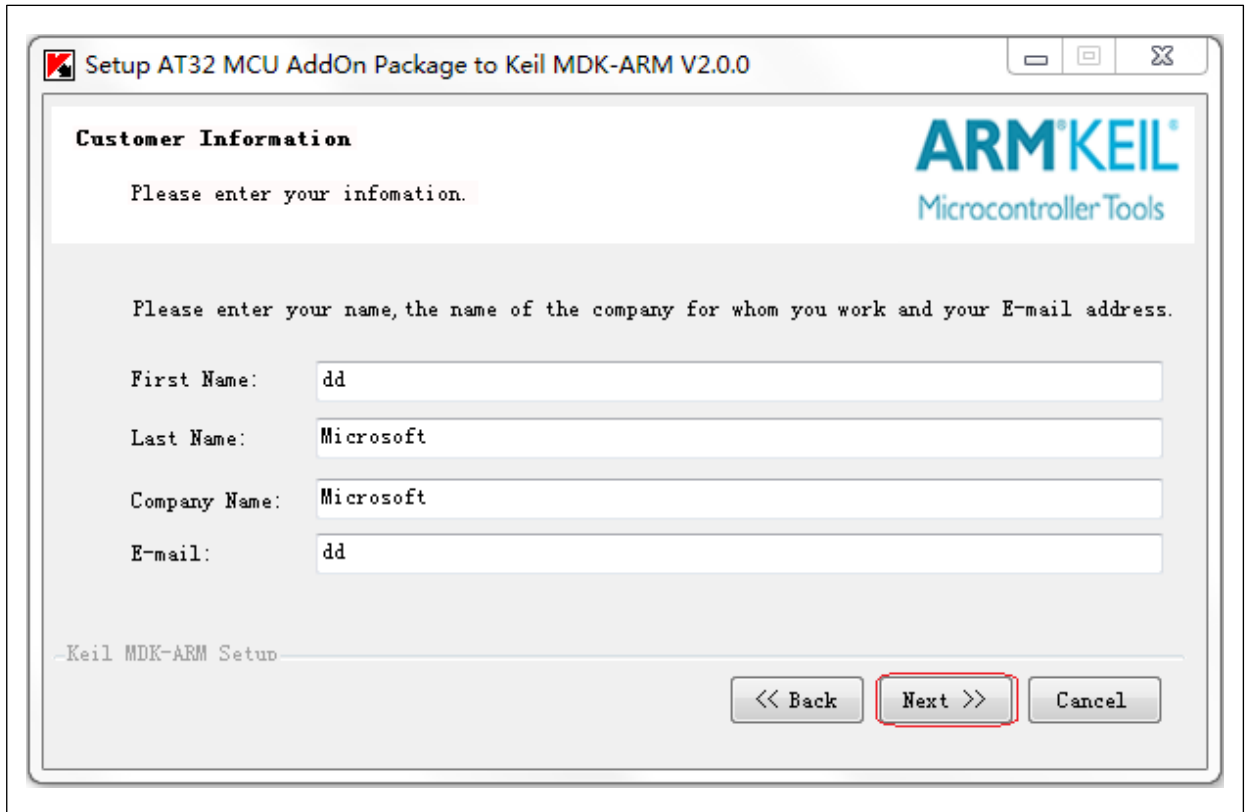
- ① Unzip *Keil4_AT32MCU_AddOn.zip*;
- ② Double click on *Keil4_AT32MCU_AddOn.exe*, and a dialog box pops up below (the specific version information is subject to the actual conditions).

Figure 6. Keil_v4 Pack installation



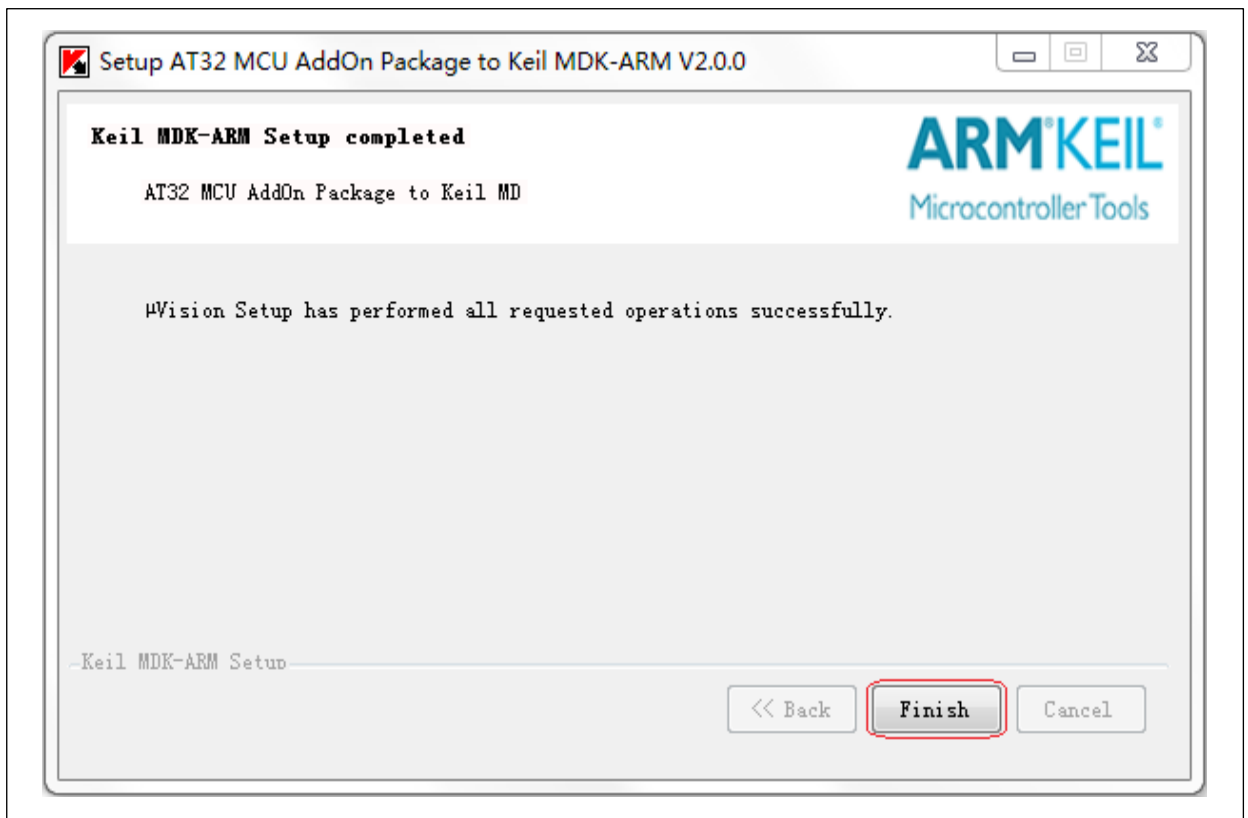
- ③ If the installation path of Keil_v4 does not match the "Destination Folder", click on "Browse" to select the actual correct path, then click on "Next", as shown below.

Figure 7. Keil_v4 Pack installation process



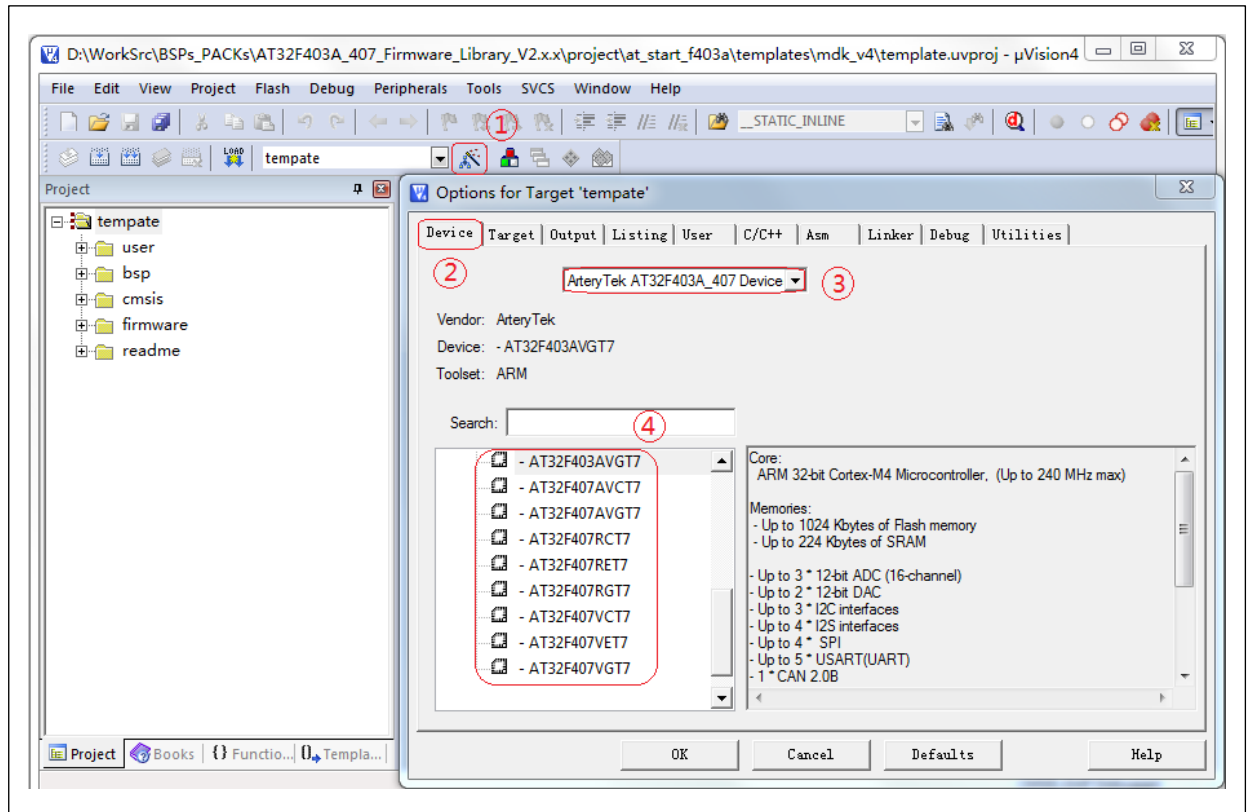
- ④ In the above “Customer Information” window, you can make some changes, but usually it is unnecessary. Then click on “Next” to start installation. The installation result is as follows.

Figure 8. Keil_v4 Pack installation complete



- ⑤ Click on “Finish”. To check whether Keil_v4 Pack is installed successfully or not, follow the below steps:
- Click on wand;
 - Select “Device”;
 - Select the desired pack file;
 - View ArteryTek-related information.

Figure 9. View Keil_v4 Pack installation status

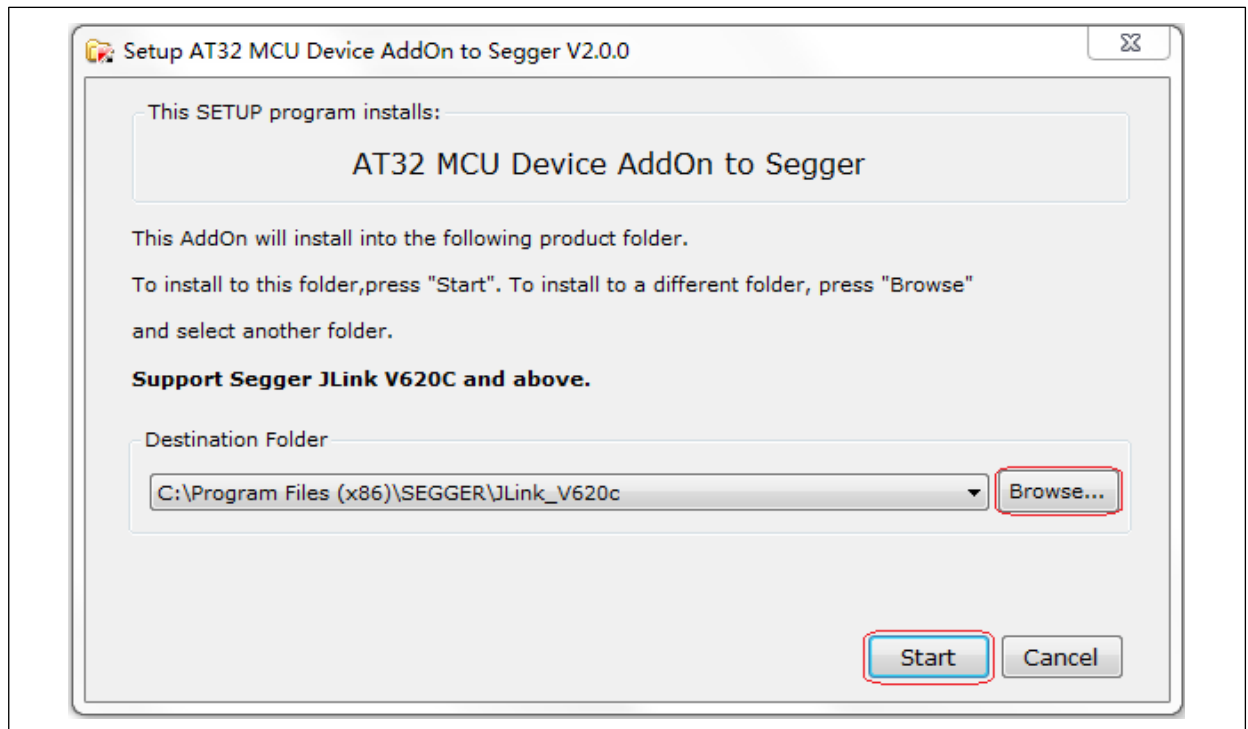


2.4 Segger Pack installation

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

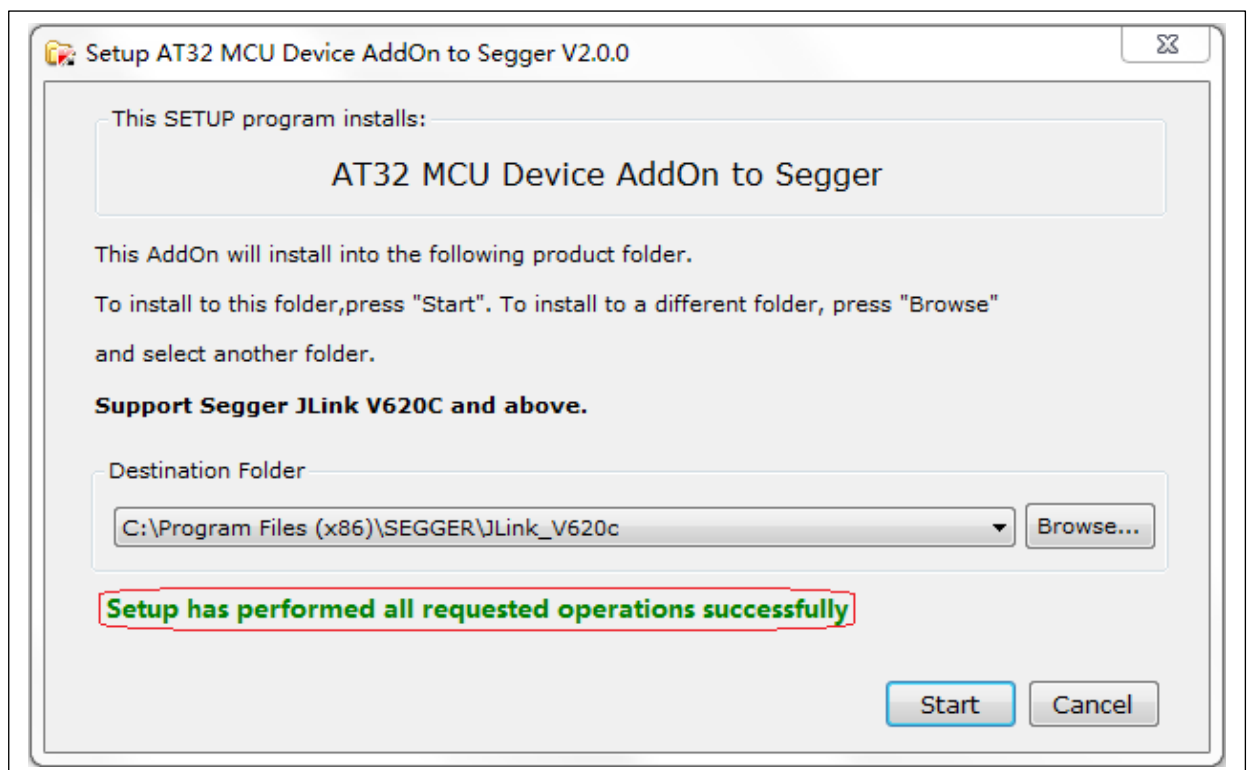
- ① Unzip *Segger_AT32MCU_AddOn.zip*;
- ② Double click on *Segger_AT32MCU_AddOn.exe*, and a dialog box pops up below (the specific version information is subject to the actual conditions).

Figure 10. Segger pack installation window



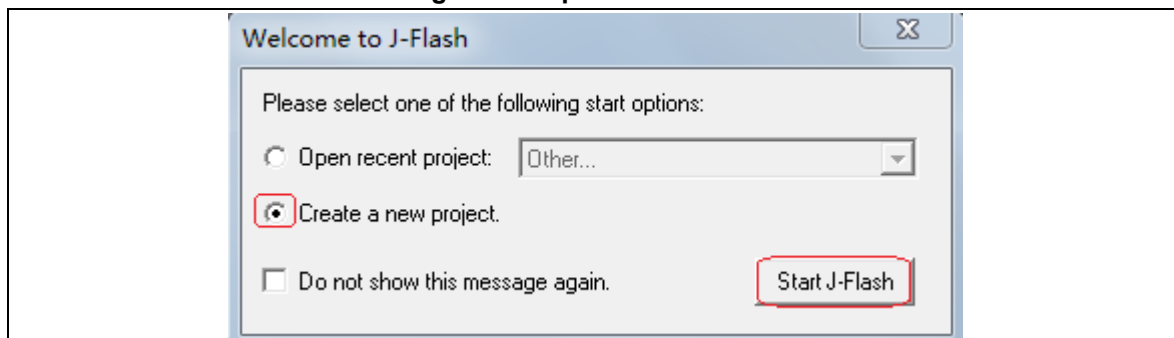
Note: If the installation path of Segger does not match the “Destination Folder”, click on “Browse” to select a correct path, then click on “Start”, as shown below.

Figure 11. Segger pack installation process



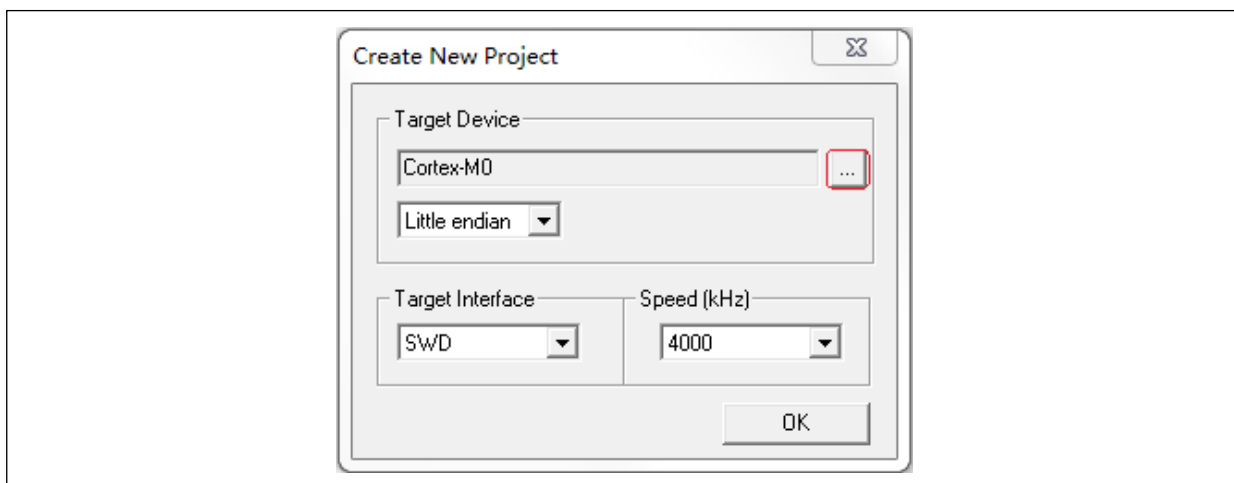
- ③ If the “Setup has performed all requested operations successfully” appears, it indicates successful installation. To check whether the installation is successful or not, follow the steps below:
- Open *J-Flash.exe*, a dialog box appears; tick “Create a new project” and click on “Start J-Flash”:

Figure 12. Open J-Flash



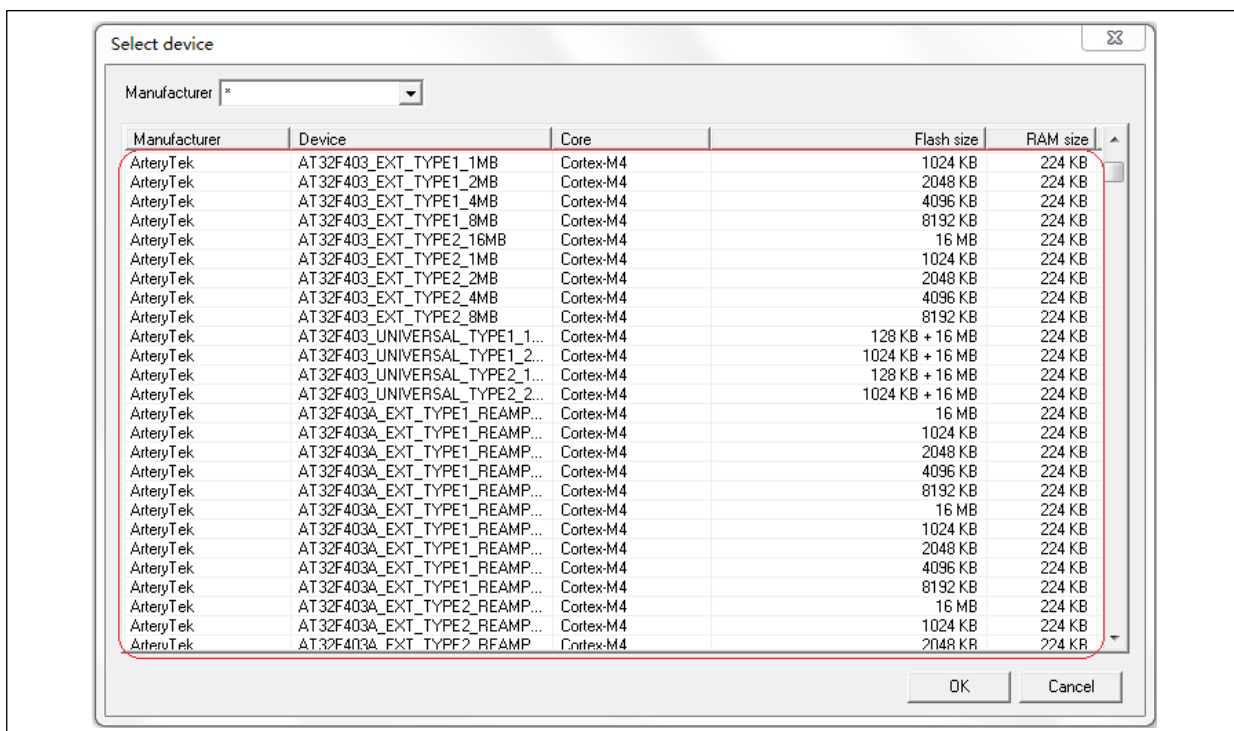
- After “Start J-Flash”, click on the check box under “Target Device”.

Figure 13. Create a new project using J-Flash



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

Figure 14. View Device information



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.

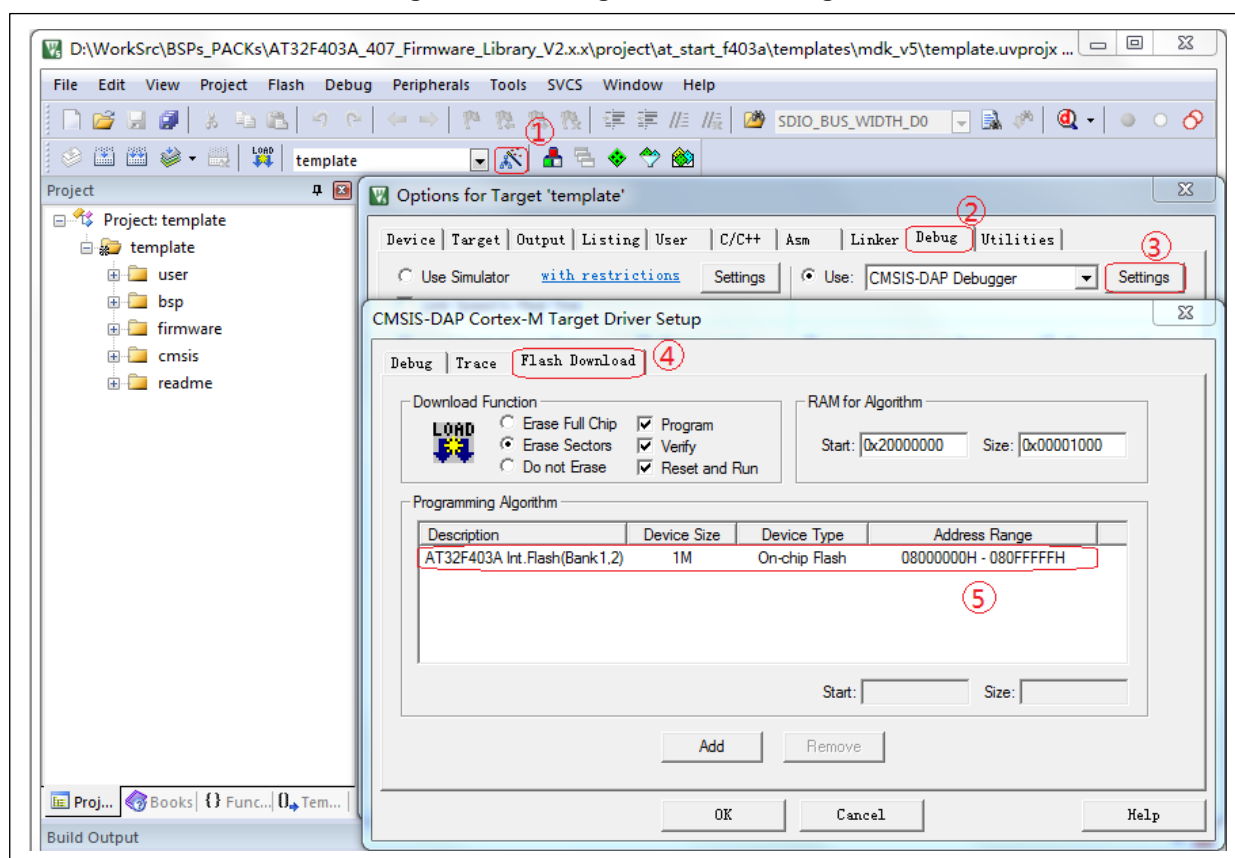
Note: AT32 MUCs have similar Flash algorithms, and this section uses AT32F403A as an example.

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.

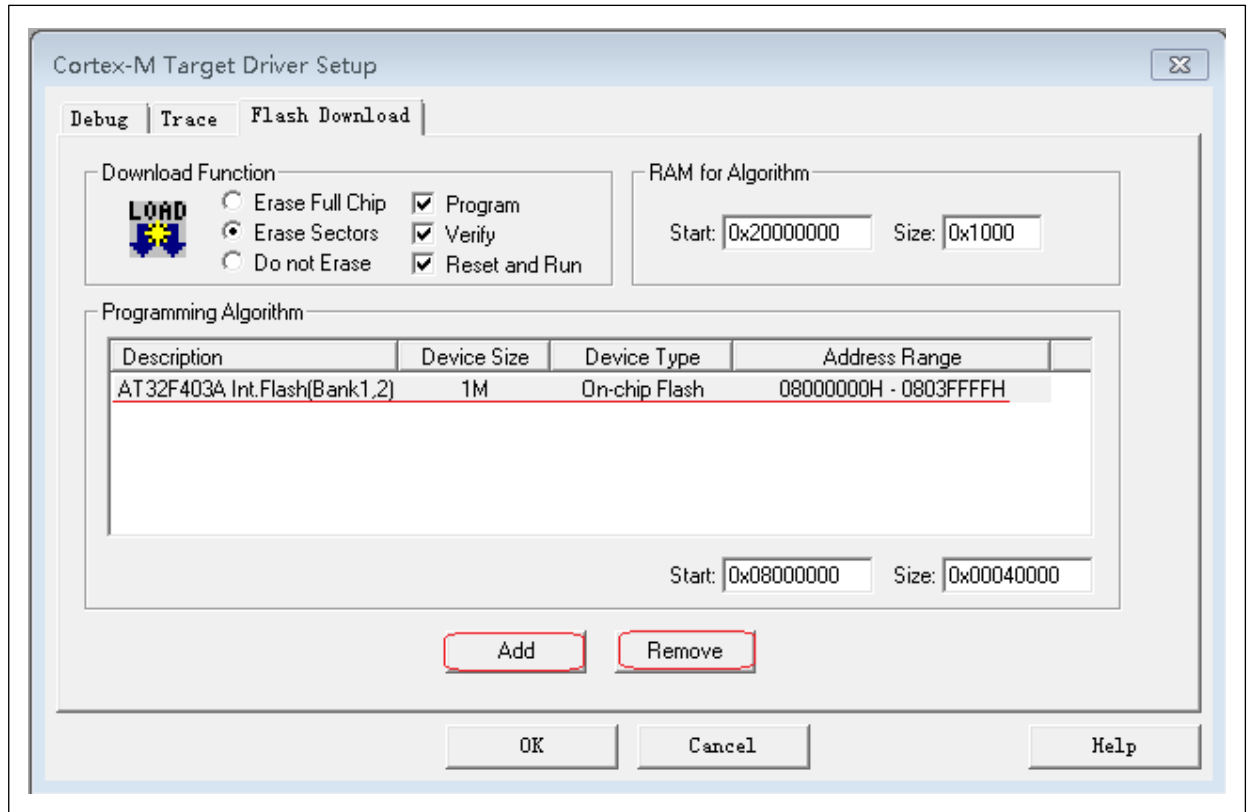
After creating a Keil IDE development tool project, the user can start Debug configuration and select the Flash algorithms. Go to *wand*→*Debug*→*Settings*→*Flash Download*, as shown below:

Figure 15. Keil algorithm file settings



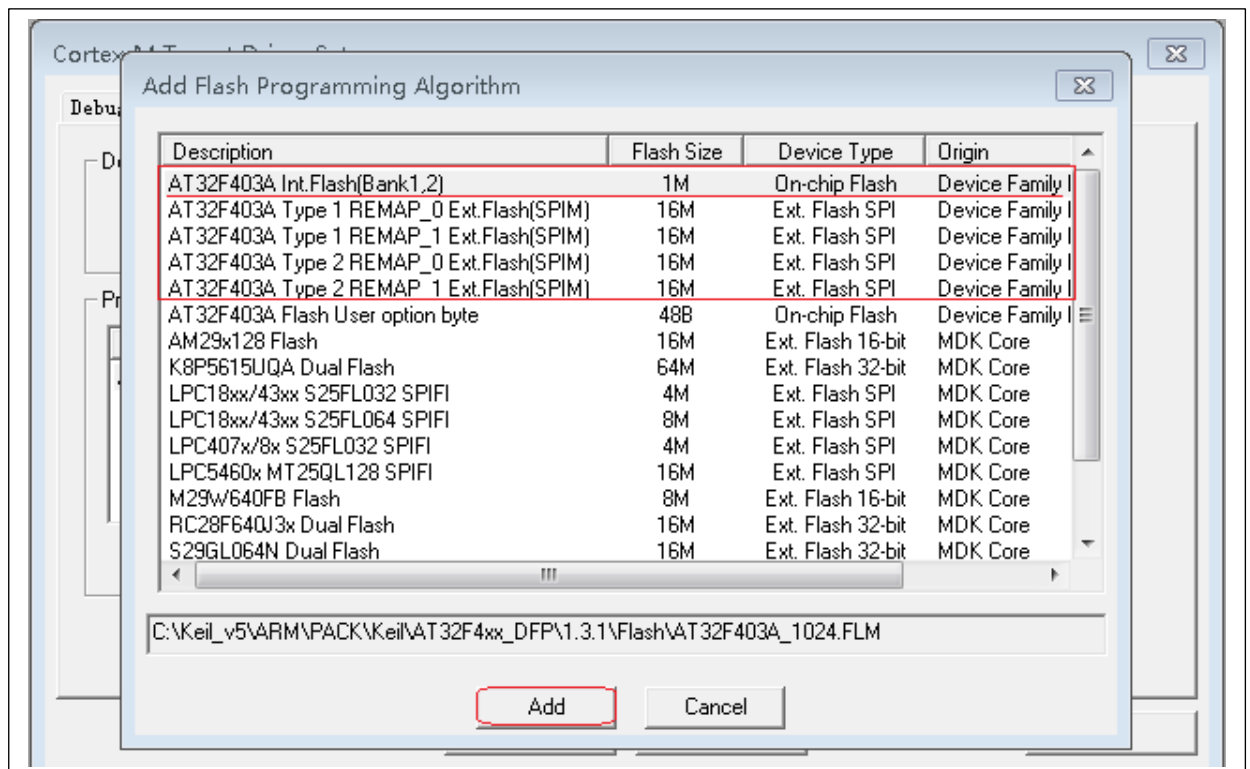
In this example, the selected Flash algorithm file is the default one. To change or remove it, click on this algorithm file, then click on *Add* or *Remove*. If the selected algorithm does not match the MCU, please follow the method below to modify.

Figure 16. Keil algorithm file configuration



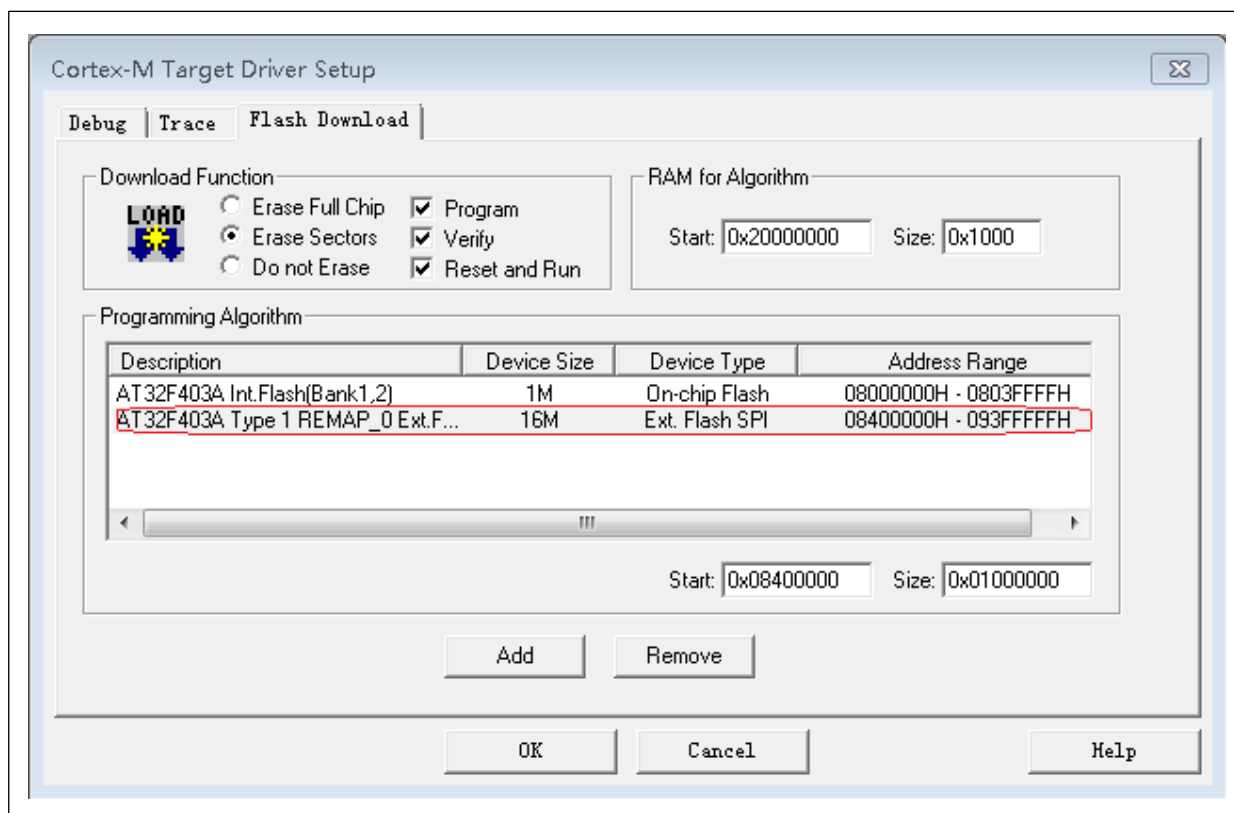
Click on *Remove* to remove the existing algorithm from the configuration, then click on *Add* to view the algorithm files associated with a MCU model and select them, as shown below:

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 SPI algorithm is added into the project.

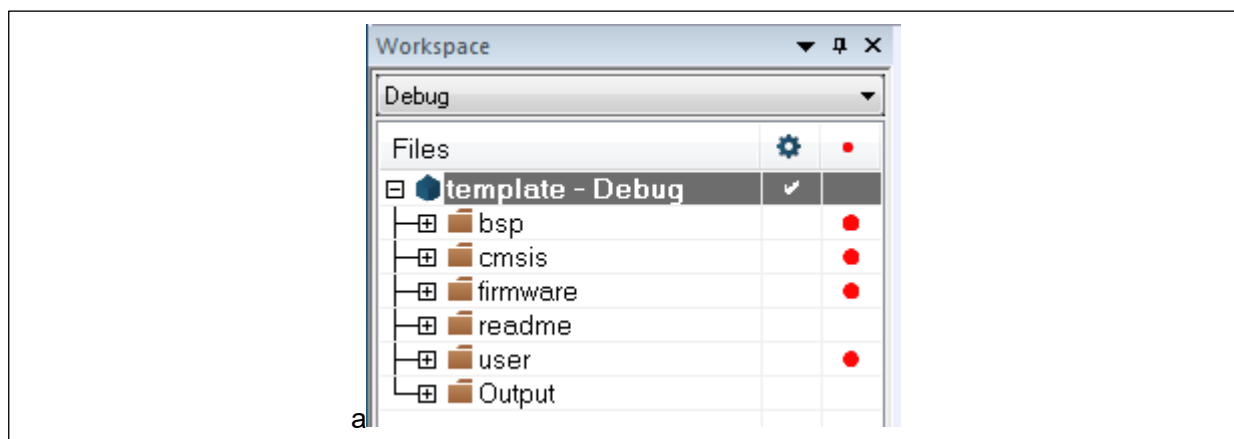
Figure 18. Add algorithm files using Keil



3.2 How to use IAR algorithm files

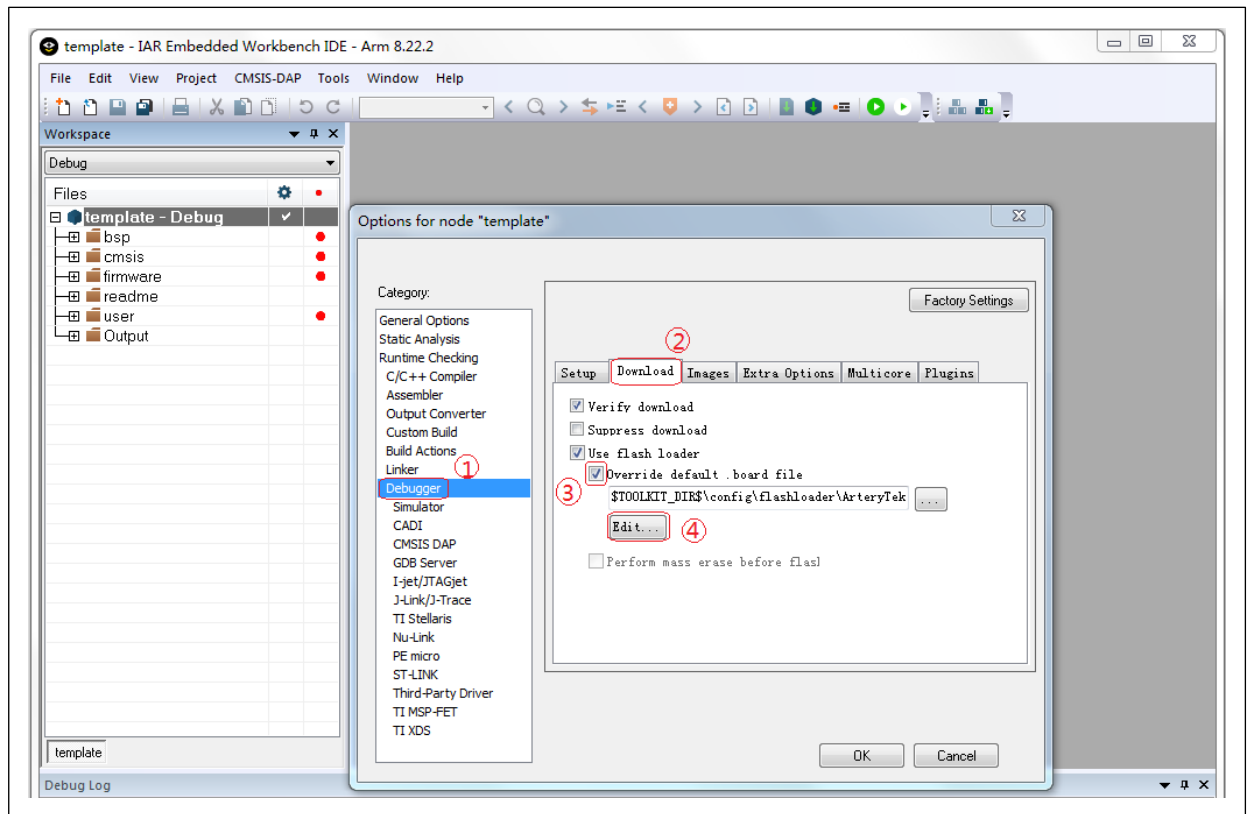
In IAR environment, the Flash algorithm files are automatically selected according to the selected MCU model during a new project configuration. To configure/modify an algorithm file manually, right-click on the file name (after an IAR project is created) in the following gray box:

Figure 19. IAR project name



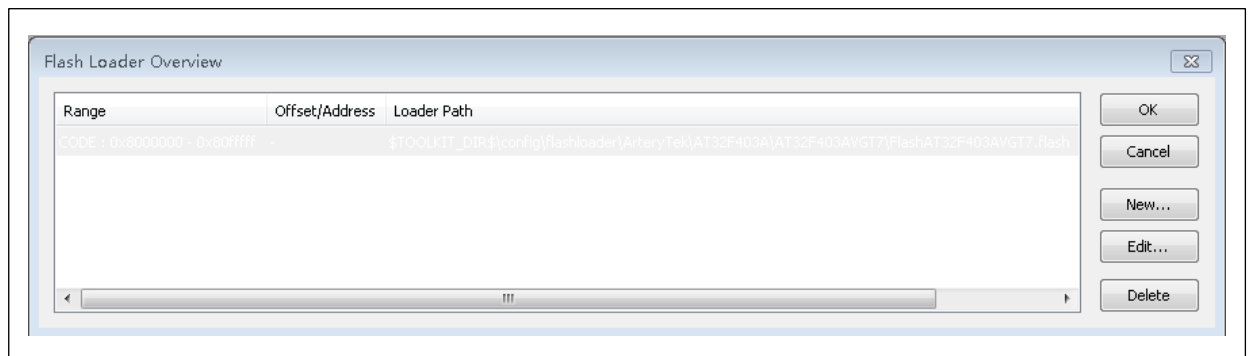
Go to *Options*—>*Debugger*—>*Download*—>Tick *Override default .board file*—>Click on *Edit*, as shown below:

Figure 20. IAR algorithm file configuration



Then the following window will be displayed.

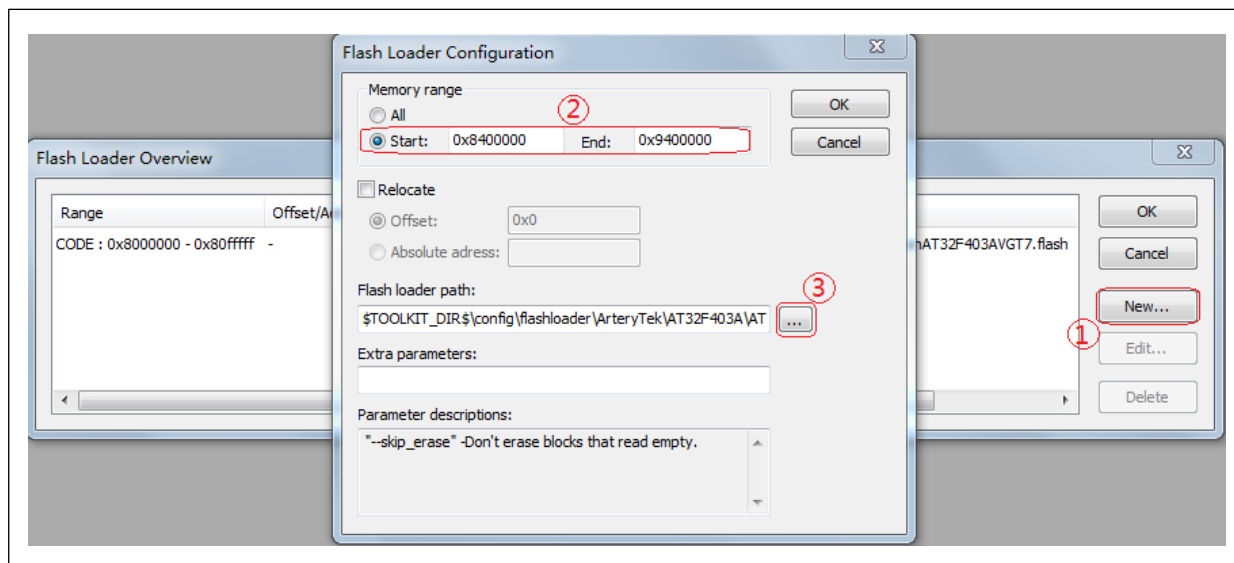
Figure 21. IAR Flash Loader overview



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

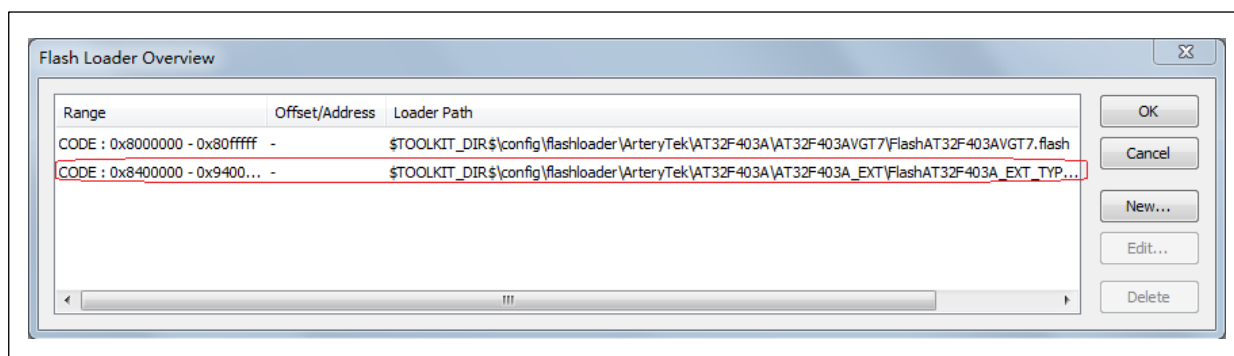
For example, click on *New*—>*Memory range*—> Select a Flash algorithm file, as shown below:

Figure 22. IAR Flash Loader configuration



This example shows how to add a SPIM Flash algorithm file. The user needs to select the corresponding MCU part number and a correct Flash algorithm file. The selected Flash algorithm configuration file is installed into IAR development environment using IAR_AT32MCU_AddOn tool. After a successful configuration, a new SPIM Flash algorithm is shown below:

Figure 23. IAR Flash Loader configuration success



1. Description of SPIM algorithms

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

Naming rules of Artery SPIM algorithm file: AT32F4xxTypeNREMAP_P Ext.Flash.

N=1,2

P=0,1

TYPEN: External SPI Flash. Select it according to the external Flash type and part number. Refer to the FLASH_SELECT register section of the corresponding MCU Reference Manual.

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

REMAP0: EXT_SPIF_GRP=000

REMAP1: EXT_SPIF_GRP=001

4 BSP introduction

4.1 Quick start

4.1.1 Template project

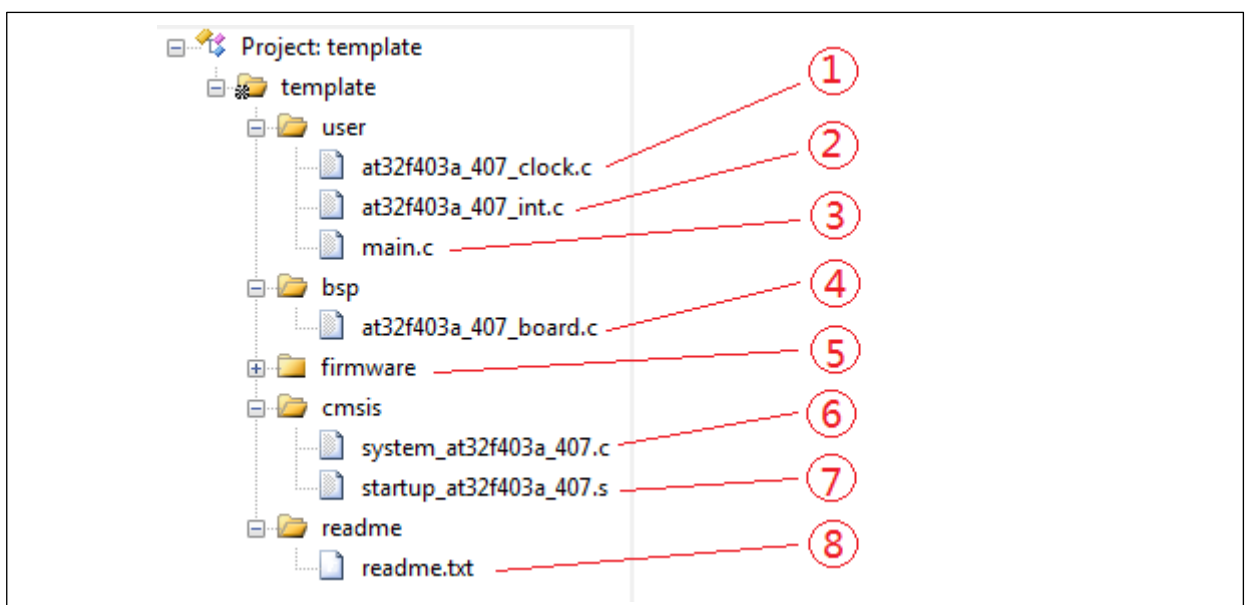
Artery firmware library BSP comes with a series of template projects built around Keil and IAR. For example, the template project of AT32F403A/407 is located in *AT32F403A_407_Firmware_Library_V2.x.x/project/at_start_xxx/templates*.

Figure 24. Template content

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

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

Figure 25. Keil_v5 template project example



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

- ① at32f403a_407_clock.c (clock configuration file) defines the default clock frequency and clock paths.
- ② at32f403a_407_int.c (interrupt file) contains some interrupt handling codes.
- ③ main.c contains the main code files.

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

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

4.1.2 BSP macro definitions

- ① To create a project, it is necessary to enable a startup code (startup_at32f403a_407.s) and open the appropriate macro definitions according to MCU part number before compiling code. Table 1 presents the correspondence between the MCU and their macro definitions.

Table 1. Summary of macro definitions

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

- ② In the header file (at32f403a_407.h), USE_STDPERIPH_DRIVER (macro definition) is used to determine whether the Keil RTE feature is used or not. Enabling this definition while Keil RTE is unused can prevent some versions of Keil-MDK from opening _RTE_ accidentally.
- ③ The configuration header file (at32f403a_407_conf.h) defines macro definitions that enable peripherals. The file can be used to control the use of peripherals. The peripherals can be disabled simply by masking _MODULE_ENABLED pertaining to peripherals, as shown below:

Figure 26. Peripheral enable macro definitions

```
#define CRM_MODULE_ENABLED
#define TMR_MODULE_ENABLED
#define RTC_MODULE_ENABLED
#define BPR_MODULE_ENABLED
#define GPIO_MODULE_ENABLED
#define I2C_MODULE_ENABLED
#define USART_MODULE_ENABLED
#define PWC_MODULE_ENABLED
#define CAN_MODULE_ENABLED
#define ADC_MODULE_ENABLED
#define DAC_MODULE_ENABLED
#define SPI_MODULE_ENABLED
#define DMA_MODULE_ENABLED
#define DEBUG_MODULE_ENABLED
#define FLASH_MODULE_ENABLED
#define CRC_MODULE_ENABLED
#define WWDT_MODULE_ENABLED
#define WDT_MODULE_ENABLED
#define EXINT_MODULE_ENABLED
#define SDIO_MODULE_ENABLED
#define XMC_MODULE_ENABLED
#define USB_MODULE_ENABLED
#define ACC_MODULE_ENABLED
#define MISC_MODULE_ENABLED
#define EMAC_MODULE_ENABLED
```

at32f403a_407_conf.h also defines the HEXT_VALUE (high-speed external clock value), which should be modified accordingly when changing an external high-speed crystal oscillator.

- ④ The system clock configuration file (*at32f403a_407_clock.c/.h*) defines the default system clock frequency and clock paths. The user, if needed, can customize the frequency multiplication process and factors, or generate corresponding clock configuration files using the clock configuration host of ArteryTek.

4.2 BSP specifications

The subsequent sections give a description of BSP specifications.

4.2.1 List of abbreviations for peripherals

Table 2. List of abbreviations for peripherals

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

4.2.2 Naming rules

The naming rules for BSP are described as follows:

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

- **Source code file**

The file name starts with “at32fxxx_ip.c”, for example, at32f403a_407_adc.c

- **Header file**

The file name starts with “at32fxxx_ip.h”, such as, at32f403a_407_adc.h

- **Constant**

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

All constants are in written in English capital letters.

- **Variable**

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

– Naming rules for functions

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

The commonly used functions are as follows:

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

4.2.3 Encoding rules

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

Type of variables:

```
typedef int32_t INT32;
```

```
typedef int16_t INT16;
```

```
typedef int8_t INT8;
```

```
typedef uint32_t UINT32;
```

```
typedef uint16_t UINT16;
```

```
typedef uint8_t UINT8;
```

```
typedef int32_t s32;
```

```
typedef int16_t s16;
```

```
typedef int8_t s8;
```

```
typedef const int32_t sc32; /*!< read only */
```

```
typedef const int16_t sc16; /*!< read only */
```

```
typedef const int8_t sc8; /*!< read only */
```

```
typedef __IO int32_t vs32;
```

```
typedef __IO int16_t vs16;
```

```
typedef __IO int8_t vs8;
```

```
typedef __I int32_t vsc32; /*!< read only */
```

```
typedef __I int16_t vsc16; /*!< read only */
```

```
typedef __I int8_t vsc8; /*!< read only */
```

```
typedef uint32_t u32;
```

```
typedef uint16_t u16;
```

```
typedef uint8_t u8;
```

```
typedef const uint32_t uc32; /*!< read only */
```

```
typedef const uint16_t uc16; /*!< read only */
```

```
typedef const uint8_t uc8; /*!< read only */
```

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

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

4.2.3.1 Flag type

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

4.2.3.2 Function status type

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

4.2.3.3 Error status type

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

4.2.3.4 Peripheral type

① Peripherals

Define the base address of peripheral in the at32fxxx_ip.h, for example, in the at32f403a_407.h:

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

Define the type of a peripheral in the at32fxxx_ip.h, for example, in the at32f403a_407_adc.h:

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

② Peripheral registers and bits

Define the type of a peripheral in the at32fxxx_ip.h, for example, in the at32f403a_407_adc.h

```
/**
 * @brief type define adc register all
 */
typedef struct
{
    /**
     * @brief adc sts register, offset:0x00
     */
    union
    {
        __IO uint32_t sts;
        struct
        {
            __IO uint32_t vmor : 1; /* [0] */
            __IO uint32_t cce : 1; /* [1] */
        };
    };
};
```

```

__IO uint32_t pcce           : 1; /* [2] */
__IO uint32_t pccs           : 1; /* [3] */
__IO uint32_t occs           : 1; /* [4] */
__IO uint32_t reserved1      : 27; /* [31:5] */
} sts_bit;
};
...
...
...
/**
 * @brief adc odt register, offset:0x4C
 */
union
{
    __IO uint32_t odt;
    struct
    {
        __IO uint32_t odt           : 16; /* [15:0] */
        __IO uint32_t adc2odt       : 16; /* [31:16] */
    } odt_bit;
};

} adc_type;

```

③ Examples of peripheral register access

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

4.3 BSP structure

4.3.1 BSP folder structure

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

Figure 27. BSP folder structure

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

Document:

- AT32Fxxx firmware library BSP&Pack user guide.pdf: refer to BSP/Pack user manual
- ReleaseNotes_AT32F403A_407_Firmware_Library.pdf: document revision history

Libraries:

- **Drivers:** driver library for peripherals
Src folder: low-level driver source file for peripherals, such as, at32fxxx_ip.c
inc folder: low-level driver header file for peripherals, such as, at32fxxx_ip.h
- **Cmsis:** Core-related files
cm0plus folder: core-related files, including cortex-m4 library, system initialization file, startup file, etc.
dsp folder: dsp-related files

Middlewares:

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

Project:

Examples: demo

Templates: template projects, including Keil4, keil5, IAR6, IAR7, IAR8 and eclipse_gcc

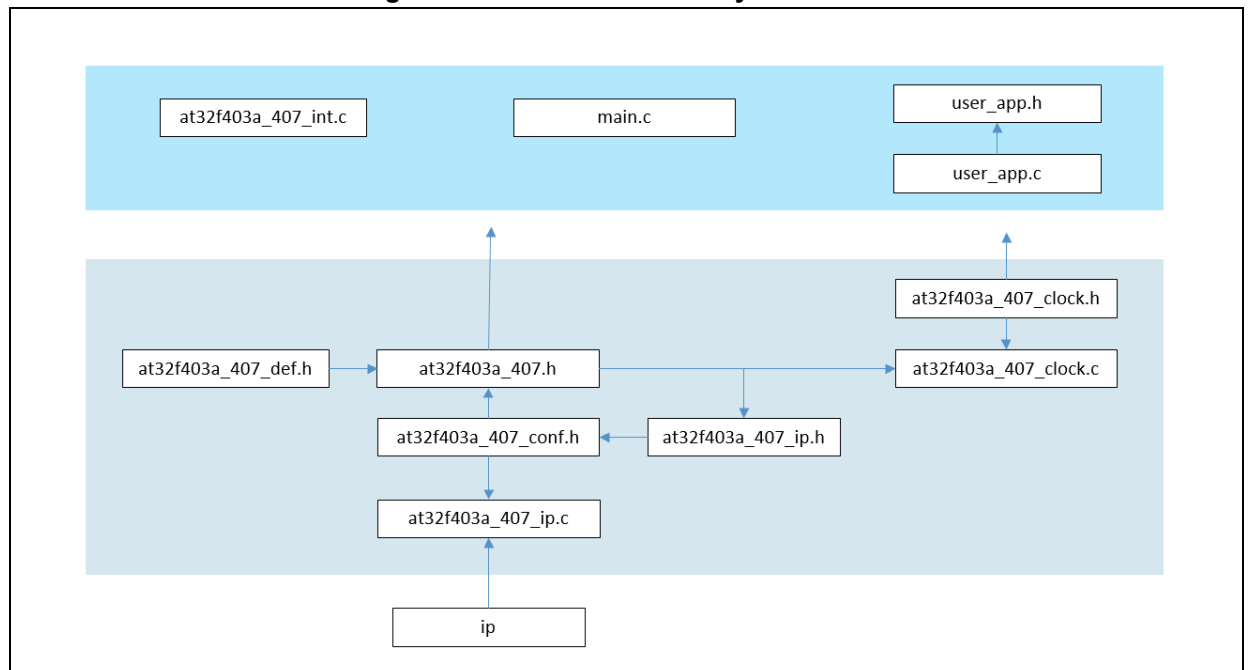
Utilities:

Store application cases

4.3.2 BSP function library structure

Figure 28 shows the architecture of BSP function library.

Figure 28. BSP function library structure



BSP function library files are described in Table 3.

Table 3. Summary of BSP function library files

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

4.3.3 Initialization and configuration for peripherals

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

GPIO initialization

Step 1: Define the gpio_init_type, for example, gpio_init_type gpio_init_struct;

Step 2: Enable GPIO clock using the function crm_periph_clock_enable;

Step 3: De-initialize the structure gpio_init_struct to allow the values of other members (mostly default values) to be correctly written, for example, gpio_default_para_init(&gpio_init_struct);

Step 4: Configure member of the structure, and write structure parameters into GPIO registers through the gpio_init, for example,

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

For more information on peripheral initialization procedure, refer to the section of peripherals of the reference manual, and the section of peripherals of the AT32Fxxx_Firmware_Library_V2.x.x.zip\project\at_start_fx\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 AT32L021 peripheral library functions

5.1 Controller area network (CAN)

CAN register structure `can_type` is defined in the “at32l021_can.h”:

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

The table below gives a list of the CAN registers.

Table 5. Summary of ADC registers

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

Register	Description
fmcfg	CAN filter mode configuration register
fscfg	CAN filter size configuration register
frf	CAN filter FIFO association register
facfg	CAN filter activate control register
fb0f1	CAN filter bank 0 filter register 1
fb0f2	CAN filter bank 0 filter register 2
fb1f1	CAN filter bank 1 filter register 1
fb1f2	CAN filter bank 1 filter register 2
...	...
Fb13f1	CAN filter bank 13 filter register 1
Fb13f2	CAN filter bank 13 filter register 2

The table below gives a list of CAN library functions.

Table 6. 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 disable/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_flag_clear	Clear the selected CAN flag

5.1.1 can_reset function

The table below describes the function can_reset.

Table 7. 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
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

Example:

```
can_reset(CAN1);
```

5.1.2 can_baudrate_default_para_init function

The table below describes the function can_baudrate_default_para_init.

Table 8. 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	The variable of the can_baudrate_type must be set first before starting.
Called functions	NA

Example:

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

5.1.3 can_baudrate_set function

The table below describes the function can_baudrate_set.

Table 9. 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 CAN baud rate
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1
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	The variable of the can_baudrate_type must be first set before starting.
Called functions	NA

The can_baudrate_type is defined in the at32l021_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);

```

5.1.4 can_default_para_init function

The table below describes the function can_default_para_init.

Table 10. 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	The variable of the can_base_type must be defined first before starting.
Called functions	NA

Example:

```

can_base_type can_base_struct;
can_default_para_init (&can_base_struct);

```

5.1.5 can_base_init function

The table below describes the function can_base_init.

Table 11. 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 parameters specified in the can_base_struct
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1
Input parameter 2	can_base_struct: can_base_type pointer
Output parameter	NA
Return value	NA
Required preconditions	The variable of the can_base_type must be defined first before starting.
Called functions	NA

The can_base_type is defined in the at32l021_can.h:

typedef struct

```

{
    can_mode_type           mode_selection;
    confirm_state            ttc_enable;
    confirm_state            aebo_enable;
    confirm_state            aed_enable;

```

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

mode_selection

Test mode selection

CAN_MODE_COMMUNICATE:	Communication mode
CAN_MODE_LOOPBACK:	Loopback mod
CAN_MODE_LISTENONLY:	Listen only mode
CAN_MODE_LISTENONLY_LOOPBACK:	Loopback + listen only mod

ttc_enable

Enable/disable time-triggered communication mode.

FALSE: Disable time-triggered communication mode

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

aebo_enable

Enable auto exit of bus-off mode.

FALSE: Automatic exit of bus-off mode is disabled

TRUE: Automatic exit of bus-off mode is enabled

aed_enable

Enable auto exit of sleep mode.

FALSE: Auto exit of sleep mode is disabled

TRUE: Auto exit of sleep mode is enabled

prsf_enable

Disable retransmission when transmit failed.

FALSE: Retransmission is enabled

TRUE: Retransmission is disabled

mdrsel_selection

Define message discard rule when reception overflows.

CAN_DISCARDING_FIRST_RECEIVED: The previous message is discarded.

CAN_DISCARDING_LAST_RECEIVED: The new incoming message is discarded.

mmssr_selection

Define multiple message transmit sequence rule.

CAN_SENDING_BY_ID: The message with the smallest identifier number is first transmitted.

CAN_SENDING_BY_REQUEST: The message with the first request order is first transmitted.

Example:

```
/* can base init */
can_base_struct.mode_selection = CAN_MODE_COMMUNICATE;
can_base_struct.ttc_enable = FALSE;
can_base_struct.aebo_enable = TRUE;
can_base_struct.aed_enable = TRUE;
can_base_struct.prsf_enable = FALSE;
can_base_struct.mdrsel_selection = CAN_DISCARDING_FIRST_RECEIVED;
can_base_struct.mmssr_selection = CAN_SENDING_BY_ID;
can_base_init(CAN1, &can_base_struct);
```

5.1.6 can_filter_default_para_init function

The table below describes the function can_filter_default_para_init.

Table 12. 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	Set the initial value for CAN filter initialization structure
Input parameter 1	can_filter_init_struct: can_filter_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	The variable of can_filter_init_type must be defined first before starting.
Called functions	NA

Example:

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

5.1.7 can_filter_init function

The table below describes the function can_filter_init.

Table 13. 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 parameters specified in the can_base_struct
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1.
Input parameter 2	can_filter_init_struct: can_filter_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	The variable of can_filter_init_type must be defined first before starting.
Called functions	NA

The can_filter_init_type is defined in the at32l021_can.h:

typedef struct

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

filter_activate_enable

Enable/disable filter bank

FALSE: Disable filter bank

TRUE: Enable filter bank

filter_mode

Select filter mode.

CAN_FILTER_MODE_ID_MASK: Identifier mask mode

CAN_FILTER_MODE_ID_LIST: Identifier list mode

filter_fifo

Select filter associated FIFO.

CAN_FILTER_FIFO0: Associated with FIFO0

CAN_FILTER_FIFO1: Associated with FIFO1

filter_number

Select filter bank.

Value range: 0~13

filter_bit

Select filter bit width

CAN_FILTER_16BIT: 16-bit

CAN_FILTER_32BIT: 32-bit

filter_id_high

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

Value range: 0x0000~0xFFFF

filter_id_low

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

Value range: 0x0000~0xFFFF

filter_mask_high

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

Value range: 0x0000~0xFFFF

filter_mask_low

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

Value range: 0x0000~0xFFFF

Example:

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

```

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

```

5.1.8 can_debug_transmission_prohibit function

The table below describes the function can_debug_transmission_prohibit.

Table 14. 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.
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);

```

5.1.9 can_ttc_mode_enable function

The table below describes the function can_ttc_mode_enable.

Table 15. 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.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```

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

```

Note: When the ttc_enable is enabled in the can_base_init, it indicates that only the time stamp is enabled (During message receive and transmit, the time stamp is captured and stored in the CAN_RFCx

and CAN_TMCx registers). But when the `can_ttc_mode_enable` is enabled, not only the time stamp is enabled, and but the time stamp transmission feature is enabled (During message transmission, the time stamp is sent on the 7th and 8th data byte).

5.1.10 can_message_transmit function

The table below describes the function `can_message_transmit`

Table 16. can_message_transmit function

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

The `can_tx_message_type` is defined in the `at32l021_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);
}
```

5.1.11 can_transmit_status_get function

The table below describes the function can_transmit_status_get.

Table 17. 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.
Input parameter 2	transmit_mailbox: indicates the mailbox number for message to send
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);
}

```

5.1.12 can_transmit_cancel function

The table below describes the function can_transmit_cancel.

Table 18. 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.
Input parameter 2	transmit_mailbox: indicates the mailbox number for message to send
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);
```

5.1.13 can_message_receive function

The table below describes the function can_message_receive.

Table 19. can_message_receive function

Name	Description
Function name	can_message_receive
Function prototype	void can_message_receive(can_type* can_x, can_rx_fifo_num_type fifo_number, can_rx_message_type* rx_message_struct);
Function description	Receive message
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1.
Input parameter 2	fifo_number: receive FIFO This parameter can be CAN_RX_FIFO0 or CAN_RX_FIFO1.
Output parameter	rx_message_struct: indicates the received message, refer to 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 at32l021_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);

```

5.1.14 can_receive_fifo_release function

The table below describes the function can_receive_fifo_release.

Table 20. 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.
Input parameter 2	fifo_number: receive FIFO number 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);
}

```

5.1.15 can_receive_message_pending_get function

The table below describes the function can_receive_message_pending_get.

Table 21. 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.
Input parameter 2	fifo_number: receive FIFO number 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);
```

5.1.16 can_operating_mode_set function

The table below describes the function can_operating_mode_set.

Table 22. 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.
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);

/* Starts communication: send and receive message */
...

```

5.1.17 can_doze_mode_enter function

The table below describes the function can_doze_mode_enter

Table 23. 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.
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);

```

5.1.18 can_doze_mode_exit function

The table below describes the function can_doze_mode_exit.

Table 24. 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.
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);
```

5.1.19 can_error_type_record_get function

The table below describes the function can_error_type_record_get.

Table 25. 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.
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);
```

5.1.20 can_receive_error_counter_get function

The table below describes the function can_receive_error_counter_get.

Table 26. 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.
Output parameter	NA
Return value	receive_error_counter: Receive error counter Value range: 0x00~0xFF
Required preconditions	NA
Called functions	NA

Example:

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

5.1.21 can_transmit_error_counter_get function

The table below describes the function can_transmit_error_counter_get.

Table 27. 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.
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);
```

5.1.22 can_interrupt_enable function

The table below describes the function can_interrupt_enable.

Table 28. 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.
Input parameter 2	can_int : Select CAN interrupts
Input parameter3	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

can_int

CAN interrupt select

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

Example:

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

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

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

5.1.23 can_flag_get function

The table below describes the function can_flag_get.

Table 29. 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.
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 value can be SET or RESET.
Required preconditions	NA
Called functions	NA

can_flag

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

- CAN_EAF_FLAG: Error active flag
- CAN_EPF_FLAG: Error passive flag
- CAN_BOF_FLAG: Bus-off flag
- CAN_ETR_FLAG: Error type record (non-zero error type flag)
- CAN_EOIF_FLAG: Error occur interrupt flag
- CAN_TM0TCF_FLAG: Mailbox 0 transmission complete flag
- CAN_TM1TCF_FLAG: Mailbox 1 transmission complete flag
- CAN_TM2TCF_FLAG: Mailbox 2 transmission complete flag
- CAN_RF0MN_FLAG: Receive FIFO0 non-empty flag
- CAN_RF0FF_FLAG: FIFO0 full flag
- CAN_RF0OF_FLAG: FIFO0 overflow flag
- CAN_RF1MN_FLAG: FIFO1 non-empty flag
- CAN_RF1FF_FLAG: FIFO1 full flag
- CAN_RF1OF_FLAG: FIFO1 overflow flag
- CAN_QDZIF_FLAG: Quit Sleep mode flag
- CAN_EDZC_FLAG: Enter Sleep mode flag
- CAN_TMEF_FLAG: Transmit mailbox empty flag (any one of three transmit mailboxes is empty)

Example:

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

5.1.24 can_interrupt_flag_get function

The table below describes the function can_interrupt_flag_get.

Table 30. 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 the selected CAN interrupt flag status
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1.
Input parameter 2	<i>can_flag</i> : indicates the selected flag Refer to can_flag
Output parameter	NA
Return value	flag_status: SET or RESET
Required preconditions	NA
Called functions	NA

can_flag:

This is used to clear the selected flag, including:

- CAN_EAF_FLAG: Error active flag
- CAN_EPF_FLAG: Error passive flag
- CAN_BOF_FLAG: Bus-off flag
- CAN_ETR_FLAG: Error type record (non-zero Error type flag)
- CAN_EOIF_FLAG: Error occur interrupt flag
- CAN_TM0TCF_FLAG: Mailbox 0 transmission complete flag
- CAN_TM1TCF_FLAG: Mailbox 1 transmission complete flag
- CAN_TM2TCF_FLAG: Mailbox 2 transmission complete flag
- CAN_RF0MN_FLAG: FIFO0 not-empty flag
- CAN_RF0FF_FLAG: FIFO0 full flag
- CAN_RF0OF_FLAG: FIFO0 overflow flag
- CAN_RF1MN_FLAG: FIFO1 not-empty flag
- CAN_RF1FF_FLAG: FIFO1 full flag
- CAN_RF1OF_FLAG: FIFO1 overflow flag
- CAN_QDZIF_FLAG: Quit Sleep mode flag
- CAN_EDZC_FLAG: Enter Sleep mode flag
- CAN_TMEF_FLAG: Transmit mailbox empty flag (any one of three transmit mailboxes is empty)

Example:

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

5.1.25 can_flag_clear function

The table below describes the function can_flag_clear.

Table 31. 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.
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);
```

5.2 CRC calculation unit (CRC)

The CRC register structure `crc_type` is defined in the “at32l021_crc.h”:

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

} crc_type;
```

The table below gives a list of the CRC registers.

Table 32. Summary of CRC registers

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

The table below gives a list of CRC library functions.

Table 33. Summary of CRC library functions

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

5.2.1 crc_data_reset function

The table below describes the function `crc_data_reset`.

Table 34. `crc_data_reset` function

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

Example:

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

5.2.2 crc_one_word_calculate function

The table below describes the function `crc_one_word_calculate`.

Table 35. `crc_one_word_calculate` function

Name	Description
Function name	<code>crc_one_word_calculate</code>
Function prototype	<code>uint32_t crc_one_word_calculate(uint32_t data);</code>
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	<code>uint32_t</code> : 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);
```

5.2.3 crc_block_calculate function

The table below describes the function `crc_block_calculate`

Table 36. `crc_block_calculate` function

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

5.2.4 crc_data_get function

The table below describes the function `crc_data_get`.

Table 37. `crc_data_get` function

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

Example:

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


5.2.5 crc_common_data_set function

The table below describes the function `crc_common_data_set`.

Table 38. `crc_common_data_set` function

Name	Description
Function name	<code>crc_common_data_set</code>
Function prototype	<code>void crc_common_data_set(uint8_t cdt_value);</code>
Function description	Configure common data register
Input parameter 1	<code>cdt_value</code> : 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);
```

5.2.6 crc_common_data_get function

The table below describes the function `crc_common_data_get`.

Table 39. `crc_common_data_get` function

Name	Description
Function name	<code>crc_common_data_get</code>
Function prototype	<code>uint8_t crc_common_data_get(void);</code>
Function description	Return the value of the command data register
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	<code>uint8_t</code> : 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 ();
```

5.2.7 crc_init_data_set function

The table below describes the function `crc_init_data_set`.

Table 40. `crc_init_data_set` function

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

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

Example:

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

5.2.8 crc_reverse_input_data_set function

The table below describes the function `crc_reverse_input_data_set`.

Table 41. `crc_reverse_input_data_set` function

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

value

Define the reverse type of input data bit.

CRC_REVERSE_INPUT_NO_AFFECTE: No effect
 CRC_REVERSE_INPUT_BY_BYTE: Byte reverse
 CRC_REVERSE_INPUT_BY_HALFWORD: Half-word reverse
 CRC_REVERSE_INPUT_BY_WORD: Word reverse

Example:

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

5.2.9 crc_reverse_output_data_set function

The table below describes the function `crc_reverse_output_data_set`.

Table 42. `crc_reverse_output_data_set` function

Name	Description
Function name	<code>crc_reverse_output_data_set</code>
Function prototype	<code>void crc_reverse_output_data_set(crc_reverse_output_type value);</code>
Function description	Define the CRC output data reverse type
Input parameter 1	value: output data bit reverse type. 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);
```

5.2.10 crc_poly_value_set function

The table below describes the function `crc_poly_value_set`.

Table 43. `crc_poly_value_set` function

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

Example:

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

5.2.11 crc_poly_value_get function

The table below describes the function `crc_poly_value_get`.

Table 44. `crc_poly_value_get` function

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

Example:

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

5.2.12 crc_poly_size_set function

The table below describes the function `crc_poly_size_set`.

Table 45. `crc_poly_size_set` function

Name	Description
Function name	<code>crc_poly_size_set</code>
Function prototype	<code>void crc_poly_size_set(crc_poly_size_type size);</code>
Function description	Set CRC polynomial valid width
Input parameter 1	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);
```

5.2.13 crc_poly_size_get function

The table below describes the function `crc_poly_size_get`.

Table 46. `crc_poly_size_get` function

Name	Description
Function name	<code>crc_poly_size_get</code>
Function prototype	<code>crc_poly_size_type crc_poly_size_get(void);</code>
Function description	Get CRC polynomial valid width
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	<code>crc_poly_size_type</code> : 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();
```

5.3 Clock and reset management (CRM)

The CRM register structure `crm_type` is defined in the “at32l021_crm.h”:

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

} crm_type;
```

The table below gives a list of the CRM registers.

Table 47. Summary of CRM registers

Register	Description
ctrl	Clock control register
cfg	Clock configuration register
clkint	Clock interrupt register
apb2rst	APB2 peripheral reset register
apb1rst	APB1 peripheral reset register
ahben	AHB peripheral clock enable register
apb2en	APB2 peripheral clock register
apb1en	APB1 peripheral clock register
bpdcc	Battery powered domain control register
ctrlsts	Control/status register
ahbrst	AHB peripheral reset register
pll	PLL configuration register
misc1	Extra register 1
piclks	Peripheral independent clock select register
misc2	Extra register 2

The table below gives a list of CRM library functions.

Table 48. Summary of CRM library functions

Function name	Description
crm_reset	Reset clock reset management register and control status
crm_lxt_bypass	Configure low-speed external clock bypass
crm_hext_bypass	Configure highed external clock bypass
crm_flag_get	Check if the selected flag is set or not
crm_hext_stable_wait	Wait HEXT to get stable
crm_hick_clock_trimming_set	High speed internal clock trimming
crm_hick_clock_calibration_set	High speed internal clock calibration
crm_periph_clock_enable	Peripheral clock enable
crm_periph_reset	Peripheral set
crm_periph_sleep_mode_clock_enable	Peripheral clock enable in Sleep mode
crm_clock_source_enable	Clock source enable
crm_flag_clear	Clear flag
crm_ertc_clock_select	ERTC clock source selection
crm_ertc_clock_enable	ERTC clock enable
crm_ahb_div_set	AHB clock division
crm_apb1_div_set	APB1 clock division
crm_apb2_div_set	APB2 clock division
crm_adc_clock_div_set	ADC 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_pll_config2	PLL clock source and frequency multiplication factor 2
crm_sysclk_switch	System clock source switch
crm_sysclk_switch_status_get	Get the status of system clock source
crm_clocks_freq_get	Get clock frequency
crm_clock_out_set	Clock output clock source
crm_interrupt_enable	Interrupt enable
crm_hick_sclk_frequency_select	Set system clock frequency as 8M or 48M when HICK is used as system clock
crm_clkout_div_set	clkout output clock division
crm_i2c1_clock_source_set	Set I2C1 peripheral clock source
crm_i2c1_clock_source_get	Get I2C1 peripheral clock source status
crm_usart1_clock_source_set	Set USART1 peripheral clock source
crm_usart1_clock_source_get	Get USART1 peripheral clock source status
crm_usart2_clock_source_set	Set USART2 peripheral clock source
crm_usart2_clock_source_get	Get USART2 peripheral clock source status

5.3.1 crm_reset function

The table below describes the function crm_reset.

Table 49. 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();
```

5.3.2 crm_lext_bypass function

The table below describes the function crm_lext_bypass.

Table 50. 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);
```


5.3.3 crm_hext_bypass function

The table below describes the function `crm_hext_bypass`.

Table 51. crm_hext_bypass function

Name	Description
Function name	<code>crm_hext_bypass</code>
Function prototype	<code>void crm_hext_bypass(confirm_state new_state);</code>
Function description	Configure high-speed external clock bypass
Input parameter 1	<code>new_state</code> : 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);
```

5.3.4 crm_flag_get function

The table below describes the function `crm_flag_get`.

Table 52. crm_flag_get function

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

flag

Select a flag to read, including:

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

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

Example:

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

5.3.5 crm_interrupt_flag_get function

The table below describes the function `crm_interrupt_flag_get`

Table 53. crm_interrupt_flag_get function

Name	Description
Function name	<code>crm_interrupt_flag_get</code>
Function prototype	<code>flag_status crm_interrupt_flag_get(uint32_t flag);</code>
Function description	Get 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: Return SET or RESET
Required preconditions	NA
Called functions	NA

lag

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)
{
}
```

5.3.6 crm_hext_stable_wait function

The table below describes the function `crm_hext_stable_wait`

Table 54. `crm_hext_stable_wait` function

Name	Description
Function name	<code>crm_hext_stable_wait</code>
Function prototype	<code>error_status crm_hext_stable_wait(void);</code>
Function description	Wait for HEXT to activate and stabilize
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	<code>error_status</code> : 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)
{
}
```

5.3.7 crm_hick_clock_trimming_set function

The table below describes the function `crm_hick_clock_trimming_set`.

Table 55. `crm_hick_clock_trimming_set` function

Name	Description
Function name	<code>crm_hick_clock_trimming_set</code>
Function prototype	<code>void crm_hick_clock_trimming_set(uint8_t trim_value);</code>
Function description	Trim HICK clock
Input parameter 1	<code>trim_value</code> : trimming value. Default value is 0x20, configurable range is from 0 to 0x3F.
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.3.8 crm_hick_clock_calibration_set function

The table below describes the function `crm_hick_clock_calibration_set`.

Table 56. crm_hick_clock_calibration_set function

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

Example:

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

5.3.9 crm_periph_clock_enable

The table below describes the function `crm_periph_clock_enable`.

Table 57. crm_periph_clock_enable function

Name	Description
Function name	<code>crm_periph_clock_enable</code>
Function prototype	<code>void crm_periph_clock_enable(crm_periph_clock_type value, confirm_state new_state);</code>
Function description	Enable peripheral clock
Input parameter 1	<code>value</code> : defines peripheral clock type
Input parameter 2	<code>new_state</code> : 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 `at32l021_crm.h`.

The naming rule of this parameter is: CRM_peripheral name_PERIPH_CLOCK.

CRM_DMA1_PERIPH_CLOCK: DMA1 peripheral clock enable

...

CRM_PWC_PERIPH_CLOCK: PWC peripheral clock enable

Example:

```
/* enable gpioa periph clock */
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
```

5.3.10 crm_periph_reset function

The table below describes the function `crm_periph_reset`.

Table 58. crm_periph_reset function

Name	Description
Function name	<code>crm_periph_reset</code>
Function prototype	<code>void crm_periph_reset(crm_periph_reset_type value, confirm_state new_state);</code>
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 `at32l021_crm.h`. The naming rule of this parameter is: CRM_peripheral name_PERIPH_RESET.

CRM_DMA1_PERIPH_RESET: DMA1 peripheral reset

...

CRM_PWC_PERIPH_RESET: PWC peripheral reset

Example:

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

5.3.11 crm_periph_sleep_mode_clock_enable function

The table below describes the function `crm_periph_sleep_mode_clock_enable`.

Table 59. crm_periph_sleep_mode_clock_enable

Name	Description
Function name	<code>crm_periph_sleep_mode_clock_enable</code>
Function prototype	<code>void crm_periph_sleep_mode_clock_enable(crm_periph_clock_sleepmd_type value, confirm_state new_state);</code>
Function description	Enable peripheral clock in Sleep mode
Input parameter 1	Value: indicates peripheral clock type in Sleep 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_clock_sleepmd_type` is defined in the `at32l021_crm.h`.

The naming rule of this parameter is: CRM_peripheral name_PERIPH_CLOCK_SLEEP_MODE.

CRM_SRAM_PERIPH_RESET: SRAM clock reset in Sleep mode

CRM_FLASH_PERIPH_RESET: Flash clock reset in Sleep mode

Example:

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

5.3.12 crm_clock_source_enable function

The table below describes the function crm_clock_source_enable function.

Table 60. 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 sources
Input parameter 1	source: Clock type
Input parameter 2	new_state: TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

source

Clock source selection.

CRM_CLOCK_SOURCE_HICK: HICK
 CRM_CLOCK_SOURCE_HEX: HEX
 CRM_CLOCK_SOURCE_PLL: PLL
 CRM_CLOCK_SOURCE_LICK: LICK

Example:

```
/* enable hex */
crm_clock_source_enable(CRM_CLOCK_SOURCE_HEX, FALSE);
```

5.3.13 crm_flag_clear function

The table below describes the function `crm_flag_clear` function.

Table 61. crm_flag_clear function

Name	Description
Function name	<code>crm_flag_clear</code>
Function prototype	<code>void crm_flag_clear(uint32_t flag);</code>
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);
```

5.3.14 crm_ertc_clock_select function

The table below describes the function `crm_ertc_clock_select` function.

Table 62. crm_ertc_clock_select function

Name	Description
Function name	<code>crm_ertc_clock_select</code>
Function prototype	<code>void crm_ertc_clock_select(crm_ertc_clock_type value);</code>
Function description	Select ERTC clock source
Input parameter 1	value: indicates ertc clock source type
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

ERTC clock source selection.

CRM_ERTC_CLOCK_NOCLK:	No clock source for ERTC
CRM_ERTC_CLOCK_LEXT:	LEXT selected as ERTC clock
CRM_ERTC_CLOCK_LICK:	LICK selected as ERTC clock
CRM_ERTC_CLOCK_HEXT_DIV:	HEXT/128 selected as ERTC clock

Example:

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


5.3.15 crm_ertc_clock_enable function

The table below describes the function `crm_ertc_clock_enable`.

Table 63. crm_ertc_clock_enable function

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

Example:

```
/* enable ertc clock */
crm_ertc_clock_enable (TRUE);
```

5.3.16 crm_ahb_div_set function

The table below describes the function `crm_ahb_div_set`.

Table 64. crm_ahb_div_set function

Name	Description
Function name	<code>crm_ahb_div_set</code>
Function prototype	<code>void crm_ahb_div_set(crm_ahb_div_type value);</code>
Function description	Configure AHB clock division
Input parameter 1	<code>value</code> : 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);
```

5.3.17 crm_apb1_div_set function

The table below describes the function `crm_apb1_div_set`.

Table 65. crm_apb1_div_set function

Name	Description
Function name	<code>crm_apb1_div_set</code>
Function prototype	<code>void crm_apb1_div_set(crm_apb1_div_type value);</code>
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);
```

5.3.18 crm_apb2_div_set function

The table below describes the function `crm_apb2_div_set`.

Table 66. crm_apb2_div_set function

Name	Description
Function name	<code>crm_apb2_div_set</code>
Function prototype	<code>void crm_apb2_div_set(crm_apb2_div_type value);</code>
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);
```

5.3.19 crm_adc_clock_div_set function

The table below describes the function `crm_adc_clock_div_set`.

Table 67. crm_adc_clock_div_set

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

value

<code>CRM_ADC_DIV_2</code> :	APB/2 used as ADC clock
<code>CRM_ADC_DIV_4</code> :	APB/4 used as ADC clock
<code>CRM_ADC_DIV_6</code> :	APB/6 used as ADC clock
<code>CRM_ADC_DIV_8</code> :	APB/8 used as ADC clock
<code>CRM_ADC_DIV_12</code> :	APB/12 used as ADC clock
<code>CRM_ADC_DIV_16</code> :	APB/16 used as ADC clock

Example:

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

5.3.20 crm_clock_failure_detection_enable function

The table below describes the function `crm_clock_failure_detection_enable`.

Table 68. crm_clock_failure_detection_enable function

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

Example:

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

5.3.21 crm_battery_powered_domain_reset function

The table below describes the function `crm_battery_powered_domain_reset`.

Table 69. crm_battery_powered_domain_reset

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

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

Example:

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

5.3.22 crm_pll_config function

The table below describes the function `crm_pll_config`.

Table 70. crm_pll_config function

Name	Description
Function name	<code>crm_pll_config</code>
Function prototype	<code>void crm_pll_config(crm_pll_clock_source_type clock_source, crm_pll_mult_type mult_value);</code>
Function description	Configure PLL clock source and frequency multiplication factor
Input parameter 1	<code>clock_source</code> : clock source for PLL frequency multiplication
Input parameter 2	<code>mult_value</code> : frequency multiplication factor
Output parameter	NA
Return value	NA
Required preconditions	PLL clock source must be enabled and stabilized before configuring and enabling PLL
Called functions	NA

clock_source

CRM_PLL_SOURCE_HICK: HICK is selected as PLL clock source

CRM_PLL_SOURCE_HEXT: HEXT is selected as PLL clock source

CRM_PLL_SOURCE_HEXT_DIV: Divided HEXT as PLL clock

mult_value

CRM_PLL_MULT_2: PLL input clock x2

CRM_PLL_MULT_3: PLL input clock x3

...

CRM_PLL_MULT_63: PLL input clock x63

CRM_PLL_MULT_64: PLL input clock x64

Example:

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

5.3.23 crm_pll_config2 function

The table below describes the function crm_pll_config2.

Table 71. crm_pll_config2 function

Name	Description
Function name	crm_pll_config2
Function prototype	void crm_pll_config2(crm_pll_clock_source_type clock_source, uint16_t pll_ns, uint16_t pll_ms, crm_pll_fr_type pll_fr);
Function description	Configure PLL clock source, frequency multiplication factor and division factor
Input parameter 1	clock_source: clock source for PLL frequency multiplication
Input parameter 2	pll_ns: frequency multiplication factor, 31~500
Input parameter 3	pll_ms: pre-division factor, 1~15
Input parameter 4	pll_fr: post-divisions factor
Output parameter	NA
Return value	NA
Required preconditions	PLL frequency multiplication clock source must be enabled and stable before enable PLL

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

Restrictions:

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

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

clock_source

CRM_PLL_SOURCE_HICK: HICK as PLL clock

CRM_PLL_SOURCE_HEXT: HEXT as PLL clock

CRM_PLL_SOURCE_HEXT_DIV: Divided HEXT as PLL clock

pll_fr

CRM_PLL_FR_1: PLL/1

CRM_PLL_FR_2: PLL/2

CRM_PLL_FR_4: PLL/4

CRM_PLL_FR_8: PLL/8

CRM_PLL_FR_16: PLL/16

CRM_PLL_FR_32: PLL/32

Example:

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

5.3.24 crm_sysclk_switch function

The table below describes the function `crm_sysclk_switch`.

Table 72. crm_sysclk_switch function

Name	Description
Function name	<code>crm_sysclk_switch</code>
Function prototype	<code>void crm_sysclk_switch(crm_sclk_type value);</code>
Function description	Switch system clock source
Input parameter 1	value: indicates the clock source for system clock
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

CRM_SCLK_HICK: HICK as system clock

CRM_SCLK_HEXT: HEXT as system clock

CRM_SCLK_PLL: PLL as system clock

Example:

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

5.3.25 crm_sysclk_switch_status_get function

The table below describes the function `crm_sysclk_switch_status_get`.

Table 73. crm_sysclk_switch_status_get function

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

Example:

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

5.3.26 crm_clocks_freq_get function

The table below describes the function `crm_clocks_freq_get`.

Table 74. crm_clocks_freq_get function

Name	Description
Function name	<code>crm_clocks_freq_get</code>
Function prototype	<code>void crm_clocks_freq_get(crm_clocks_freq_type *clocks_struct);</code>
Function description	Get clock frequency
Input parameter 1	<code>clocks_struct</code> : <code>crm_clocks_freq_type</code> pointer, including clock frequency
Input parameter 2	NA
Output parameter	NA
Return value	<code>crm_sclk_type</code> : 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 `at32l021_crm.h`:

typedef struct

{

uint32_t sclk_freq;

uint32_t ahb_freq;

uint32_t apb2_freq;

uint32_t apb1_freq;

uint32_t adc_freq;

} `crm_clocks_freq_type`;

sclk_freq

Get the system clock frequency, in Hz

ahb_freq

Get the clock frequency of AHB, in Hz

apb2_freq

Get the clock frequency of APB2, in Hz

apb1_freq

Get the clock frequency of APB1, in Hz

adc_freq

Get the clock frequency of ADC, in Hz

Example:

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

5.3.27 crm_clock_out_set function

The table below describes the function `crm_clock_out_set`.

Table 75. crm_clock_out_set function

Name	Description
Function name	<code>crm_clock_out_set</code>
Function prototype	<code>void crm_clock_out_set(crm_clkout_select_type clkout);</code>
Function description	Select clock source output on clkout pin
Input parameter 1	clkout: clock source output on clkout pin
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.3.28 crm_interrupt_enable function

The table below describes the function `crm_interrupt_enable`.

Table 76. crm_interrupt_enable function

Name	Description
Function name	<code>crm_interrupt_enable</code>
Function prototype	<code>void crm_interrupt_enable(uint32_t crm_int, confirm_state new_state);</code>
Function description	Enable interrupts
Input parameter 1	crm_int: indicates the selected interrupt
Input parameter 2	new_state: Enable (TRUE), disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

crm_int

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

Example:

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


5.3.29 crm_hick_sclk_frequency_select function

The table below describes the function `crm_hick_sclk_frequency_select`.

Table 77. `crm_hick_sclk_frequency_select` function

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

value

CRM_HICK_SCLK_8MHZ: 8MHz HICK used as system clock

CRM_HICK_SCLK_48MHZ: 48MHz HICK used as system clock

Example:

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

5.3.30 crm_clkout_div_set function

The table below describes the function `crm_clkout_div_set`.

Table 78. crm_clkout_div_set

Name	Description
Function name	<code>crm_clkout_div_set</code>
Function prototype	<code>void crm_clkout_div_set(crm_clkout_div_type clkout_div);</code>
Function description	Configure clkout output clock division
Input parameter 1	<code>clkout_div</code> : indicates the clkout output frequency division factor
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

value

<code>CRM_CLKOUT_DIV_1</code> :	CLKOUT divided by 1
<code>CRM_CLKOUT_DIV_2</code> :	CLKOUT divided by 2
<code>CRM_CLKOUT_DIV_4</code> :	CLKOUT divided by 4
<code>CRM_CLKOUT_DIV_8</code> :	CLKOUT divided by 8
<code>CRM_CLKOUT_DIV_16</code> :	CLKOUT divided by 16
<code>CRM_CLKOUT_DIV_64</code> :	CLKOUT divided by 64
<code>CRM_CLKOUT_DIV_128</code> :	CLKOUT divided by 128
<code>CRM_CLKOUT_DIV_256</code> :	CLKOUT divided by 256
<code>CRM_CLKOUT_DIV_512</code> :	CLKOUT divided by 512

Example:

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

5.3.31 crm_usart_clock_select function

The table below describes the function `usart_clock_select`.

Table 79. crm_usart_clock_select function

Name	Description
Function name	<code>crm_usart_clock_select</code>
Function prototype	<code>void crm_usart_clock_select(crm_usart_index_type usart_index, crm_usart_clock_source_type value);</code>
Function description	Serial interface peripheral clock source select
Input parameter 1	<code>usart_index</code> : the index of the USART peripheral
Input parameter 2	<code>value</code> : clock source type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

usart_index

<code>CRM_USART1</code> :	USART 1
<code>CRM_USART2</code> :	USART 2

value

CRM_USART_CLOCK_SOURCE_PCLK: PCLK as USART clock source
 CRM_USART_CLOCK_SOURCE_SCLK: System clock as USART clock source
 CRM_USART_CLOCK_SOURCE_HICK48: HICK48 as USART clock source
 CRM_USART_CLOCK_SOURCE_LEXT: LEXT as USART clock source

Example:

```
/* select pclk as usart1 clock */
crm_usart_clock_select (CRM_USART1, CRM_USART_CLOCK_SOURCE_PCLK);
```

5.3.32 crm_usart_clock_get function

The table below describes the function crm_usart_clock_get.

Table 80. crm_usart_clock_get function

Name	Description
Function name	crm_usart_clock_get
Function prototype	crm_usart_clock_source_type crm_usart_clock_get(crm_usart_index_type usart_index);
Function description	Get serial interface peripheral clock source status
Input parameter 1	usart_index: the index of the USART peripheral
Input parameter 2	NA
Output parameter	NA
Return value	crm_usart_clock_source_type: USART clock source type
Required preconditions	NA
Called functions	NA

usart_index

CRM_USART1: USART 1
 CRM_USART2: USART 2

Example:

```
/* get usart1 clock */
crm_usart_clock_source_type usart_clock;
usart_clock = crm_usart_clock_get (CRM_USART1);
```

5.3.33 crm_crm_i2c_clock_select function

The table below describes the function `crm_i2c_clock_select`.

Table 81. crm_i2c_clock_select function

Name	Description
Function name	<code>crm_i2c_clock_select</code>
Function prototype	<code>void crm_i2c_clock_select(crm_i2c_type i2c_index, crm_i2c_clock_source_type value);</code>
Function description	I2C peripheral clock source select
Input parameter 1	<code>i2c_index</code> : the index of the I2C peripheral
Input parameter 2	Value: clock source type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

i2c_index

CRM_I2C1: I2C1

value

CRM_I2C_CLOCK_SOURCE_PCLK: PCLK as I2C clock source

CRM_I2C_CLOCK_SOURCE_SCLK: SCLK as I2C clock source

CRM_I2C_CLOCK_SOURCE_HICK48: HICK48 as I2C clock source

Example:

```
/* select pclk as i2c1 clock */  
crm_i2c_clock_select (CRM_I2C1, CRM_I2C_CLOCK_SOURCE_PCLK);
```

5.3.34 crm_crm_i2c_clock_get function

The table below describes the function `crm_i2c_clock_get`.

Table 82. `crm_i2c_clock_get` function

Name	Description
Function name	<code>crm_i2c_clock_get</code>
Function prototype	<code>crm_i2c_clock_source_type crm_i2c_clock_get(crm_i2c_type i2c_index);</code>
Function description	Get I2C peripheral clock source
Input parameter 1	<code>i2c_index</code> : the index of the I2C peripheral
Input parameter 2	NA
Output parameter	NA
Return value	<code>crm_i2c_clock_source_type</code> : I2C clock source type
Required preconditions	NA
Called functions	NA

i2c_index

CRM_I2C1: I2C1

Example:

```
/* get i2c1 clock */  
crm_i2c_clock_source_type i2c_clock;  
i2c_clock = crm_i2c_clock_get (CRM_I2C1);
```

5.4 Debug

The DEBUG register structure debug_type is defined in the “at32L021_debug.h”:

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

} debug_type;
```

The table below gives a list of the DEBUG registers.

Table 83. Summary of DEBUG registers

Register	Description
idcode	Device ID
ctrl	Control register

The table below gives a list of DEBUG library functions.

Table 84. Summary of DEBUG library functions

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

5.4.1 debug_device_id_get function

The table below describes the function debug_device_id_get.

Table 85. 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();
```

5.4.2 debug_device_series_id_get function

The table below describes the function debug_device_series_id_get.

Table 86. debug_device_series_id_getfunction

Name	Description
Function name	debug_device_series_id_get
Function prototype	uint32_t debug_device_series_id_get(void);
Function description	Read device series ID
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	Return 32-bit series ID
Required preconditions	NA
Called functions	NA

Example:

```
/* get idcode */
uint32_t series_id= 0;
series_id = debug_device_series_id_get ();
```

5.4.3 debug_periph_mode_set function

The table below describes the function debug_periph_mode_set.

Table 87. 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	Set debug mode for the selected peripheral or mode
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

Set debug mode for the selected peripheral or low-power mode.

DEBUG_SLEEP:	DEBUG in SLEEP mode
DEBUG_DEEPSLEEP:	DEBUG in DEEPSLEEP mode
DEBUG_STANDBY:	DEBUG in STANDBY mode
DEBUG_CAN1_PAUSE:	CAN1 pause in debug mode
DEBUG_WDT_PAUSE:	WDT pause in debug mode
DEBUG_WWDT_PAUSE:	WWDT pause in debug mode
DEBUG_TMR1_PAUSE:	TMR1 pause in debug mode
DEBUG_TMR3_PAUSE:	TMR3 pause in debug mode
DEBUG_TMR6_PAUSE:	TMR6 pause in debug mode
DEBUG_TMR14_PAUSE:	TMR14 pause in debug mode
DEBUG_TMR15_PAUSE:	TMR15 pause in debug mode
DEBUG_TMR16_PAUSE:	TMR16 pause in debug mode
DEBUG_TMR17_PAUSE:	TMR17 pause in debug mode
DEBUG_I2C1_SMBUS_TIMEOUT:	I2C1 SMBUS TIMEOUT in debug mode
DEBUG_I2C2_SMBUS_TIMEOUT:	I2C2 SMBUS TIMEOUT in debug mode
DEBUG_ERTC_PAUSE:	ERTC pause in debug mode
DEBUG_ERTC_512_PAUSE:	ERTC 512Hz pause in debug mode

Example:

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


5.5 DMA controller

The DMA register structure `dma_type` is defined in the “at32l021_dma.h”:

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

} dma_type;
```

DMA channel register structure `dma_channel_type` is defined in the “at32l021_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 88. Summary of DMA registers

Register	Description
<code>dma_sts</code>	DMA status register
<code>dma_clr</code>	DMA status clear register
<code>dma_c1ctrl</code>	DMA channel 1 configuration register
<code>dma_c1dtcnt</code>	DMA channel 1 number of data register
<code>dma_c1paddr</code>	DMA channel 1 peripheral address register
<code>dma_c1maddr</code>	DMA channel 1 memory address register
<code>dma_c2ctrl</code>	DMA channel 2 configuration register
<code>dma_c2dtcnt</code>	DMA channel 2 number of data register
<code>dma_c2paddr</code>	DMA channel 2 peripheral address register
<code>dma_c2maddr</code>	DMA channel 2 memory address register
<code>dma_c3ctrl</code>	DMA channel 3 configuration register
<code>dma_c3dtcnt</code>	DMA channel 3 number of data register
<code>dma_c3paddr</code>	DMA channel 3 peripheral address register
<code>dma_c3maddr</code>	DMA channel 3 memory address register
<code>dma_c4ctrl</code>	DMA channel 4 configuration register
<code>dma_c4dtcnt</code>	DMA channel 4 number of data register
<code>dma_c4paddr</code>	DMA channel 4 peripheral address register
<code>dma_c4maddr</code>	DMA channel 4 memory address register
<code>dma_c5ctrl</code>	DMA channel 5 configuration register
<code>dma_c5dtcnt</code>	DMA channel 5 number of data register

Register	Description
dma_c5paddr	DMA channel 5 peripheral address register
dma_c5maddr	DMA channel 5 memory address register
dma_src_sel0	Channel source register 0
dma_src_sel1	Channel source register 1

The table below gives a list of DMA library functions.

Table 89. Summary of DMA library functions

Function name	Description
dma_default_para_init	Initialize the parameters of the dma_init_struct
dma_init	Initialize the selected DMA channel
dma_reset	Reset the selected DMA channel
dma_data_number_set	Set the number of data transfer of a given channel
dma_data_number_get	Get the number of data transfer of a given channel
dma_interrupt_enable	Enable DMA channel interrupt
dma_channel_enable	Enable DMA channel
dma_flexible_config	Configure flexible DMA request mapping
dma_flag_get	Get the flag of DMA channels
dma_flag_clear	Clear the flag of DMA channels

5.5.1 dma_default_para_init function

The table below describes the function dma_default_para_init.

Table 90. dma_default_para_init function

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

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

Table 91. 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

Member	Default values
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);
```

5.5.2 dma_init function

The table below describes the function dma_init.

Table 92. dma_init function

Name	Description
Function name	dma_init
Function prototype	void dma_init(dma_channel_type* dma_channel, dma_init_type* dma_init_struct)
Function description	Initialize the selected DMA channel
Input parameter 1	dma_channel: DMA_CHANNEL defines a DMA channel number, x=1, y=1...5
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 at32l021_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};
/* dma1 channel1 configuration */
dma_init_struct.buffer_size = BUFFER_SIZE;
dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;
dma_init_struct.memory_base_addr = (uint32_t)src_buffer;
dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_HALFWORD;
dma_init_struct.memory_inc_enable = TRUE;
dma_init_struct.peripheral_base_addr = (uint32_t)0x4001100C;
dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_HALFWORD;
dma_init_struct.peripheral_inc_enable = FALSE;
dma_init_struct.priority = DMA_PRIORITY_MEDIUM;
dma_init_struct.loop_mode_enable = FALSE;
```

```
dma_init(DMA1_CHANNEL1, &dma_init_struct);
```

5.5.3 dma_reset function

The table below describes the function dma_reset.

Table 93. dma_reset function

Name	Description
Function name	dma_reset
Function prototype	void dma_reset(dma_channel_type* dma_channel);
Function description	Reset the selected DMA channel
Input parameter 1	dma_channel: DMAx_CHANNELy defines a DMA channel number, x=1, y=1...5
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.5.4 dma_data_number_set function

The table below describes the function dma_data_number_set.

Table 94. dma_data_number_set function

Name	Description
Function name	dma_data_number_set
Function prototype	void dma_data_number_set(dma_channel_type* dma_channel, uint16_t data_number);
Function description	Set the number of data transfer of the selected DMA channel
Input parameter 1	dma_channel: DMAx_CHANNELy defines a DMA channel number, x=1, y=1...5
Input parameter 2	data_number: indicates the number of data transfer, up to 65535
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set dma1 channel1 data count is 0x100*/
dma_data_number_set(DMA1_CHANNEL1, 0x100);
```

5.5.5 dma_data_number_get function

The table below describes the function dma_data_number_get.

Table 95. dma_data_number_get function

Name	Description
Function name	dma_data_number_get
Function prototype	uint16_t dma_data_number_get(dma_channel_type* dma_channel);
Function description	Get the number of data transfer of the selected DMA channel
Input parameter 1	dma_channel: DMA_CHANNEL defines a DMA channel number, x=1, y=1...5
Output parameter	NA
Return value	Get the number of data transfer of a DMA channel
Required preconditions	NA
Called functions	NA

Example:

```
/* get dma1 channel1 data count*/
uint16_t data_counter;
data_counter = dma_data_number_get(DMA1_CHANNEL1);
```

5.5.6 dma_interrupt_enable function

The table below describes the function dma_interrupt_enable.

Table 96. dma_interrupt_enable function

Name	Description
Function name	dma_interrupt_enable
Function prototype	void dma_interrupt_enable(dma_channel_type* dma_channel, uint32_t dma_int, confirm_state new_state);
Function description	Enable DMA channels interrupt
Input parameter 1	dma_channel: DMA_CHANNEL defines a DMA channel number, x=1, y=1...5
Input parameter 2	dma_int: interrupt source selection
Input parameter3	new_state: interrupt enable/disable
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dma_int

Select DMA interrupt source

DMA_FDT_INT: Transfer complete interrupt

DMA_HDT_INT: Half transfer complete interrupt

DMA_DTERR_INT: Transfer error interrupt

new_state

Enable or disable DMA channel interrupt

FALSE: Disabled

TRUE: Enabled

Example:

```
/* enable dma1 channel1 transfer full data interrupt */
dma_interrupt_enable(DMA1_CHANNEL1, DMA_FDT_INT, TRUE);
```

5.5.7 dma_channel_enable function

The table below describes the function dma_channel_enable.

Table 97. dma_channel_enable function

Name	Description
Function name	dma_channel_enable
Function prototype	void dma_channel_enable(dma_channel_type* dma_channel, confirm_state new_state);
Function description	Enable the selected DMA channel
Input parameter 1	dma_channel: DMAx_CHANNELy defines a DMA channel number, x=1, y=1...5
Input parameter 2	new_state: Enable or disable the selected DMA channel
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

Enable or disable DMA channels

FALSE: Disabled

TRUE: Enabled

Example:

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

5.5.8 dma_flexible_config function

The table below describes the function dma_flexible_enable.

Table 98. dma_flexible_config function

Name	Description
Function name	dma_flexible_config
Function prototype	void dma_flexible_config(dma_type* dma_x, uint8_t flex_channelx, dma_flexible_request_type flexible_request);
Function description	Configure DMAMUX
Input parameter 1	dma_x: DMAx, x=1
Input parameter 2	flex_channelx: FLEX_CHANNELx channel number, x=1...5
Input parameter3	flexible_request: DMAMUX channel request ID
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flexible_request

Table below shows the flexible mapping request source ID:

Table 99. DMAMUX channel request source ID

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

Example:

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


5.5.9 dma_flag_get function

The table below describes the function dma_flag_get.

Table 100. 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

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);
}
```

5.5.10 dma_interrupt_get function

The table below describes the function dma_interrupt_flag_get.

Table 101. dma_interrupt_flag_get function

Name	Description
Function name	dma_interrupt_flag_get
Function prototype	flag_status dma_interrupt_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

flag_status

RESET: Flag is reset

SET: Flag is set

Example:

```
if(dma_interrupt_flag_get (DMA1_FDT1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
}
```

5.5.11 dma_flag_clear function

The table below describes the function dma_flag_clear.

Table 102. 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

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

Example:

```
if(dma_flag_get(DMA1_FDT1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
    dma_flag_clear(DMA1_FDT1_FLAG);
}
```

5.6 Real-time clock (ERTC)

The ERTC register structure ertc_type is defined in the “at32l021_ertc.h”:

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

} ertc_type;
```

The table below gives a list of the ERTC registers:

Table 103. Summary of ERTC registers

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

The table below gives a list of ERTC library functions.

Table 104. Summary of ERTC library functions

Function name	Description
ertc_num_to_bcd	Convert number to BCD code
ertc_bcd_to_num	Convert BCD code to number
ertc_write_protect_enable	Enable write protection
ertc_write_protect_disable	Disable write protection
ertc_wait_update	Wait for register update complete
ertc_wait_flag	Wait flag
ertc_init_mode_enter	Enter initialization mode
ertc_init_mode_exit	Exit initialization mode
ertc_reset	Reset ERTC registers
ertc_divider_set	Divider setting
ertc_hour_mode_set	Hour mode setting
ertc_date_set	Date setting
ertc_time_set	Time setting
ertc_calendar_get	Get calendar
ertc_sub_second_get	Get the current subsecond
ertc_alarm_mask_set	Set alarm mask
ertc_alarm_week_date_select	Alarm time format selection (week/date)
ertc_alarm_set	Set alarm
ertc_alarm_sub_second_set	Set alarm subsecond
ertc_alarm_enable	Enable alarm
ertc_alarm_get	Get alarm value
ertc_alarm_sub_second_get	Get alarm subsecond
ertc_wakeup_clock_set	Select wakeup clock source
ertc_wakeup_counter_set	Set wakeup counter value
ertc_wakeup_counter_get	Get wakeup counter value
ertc_wakeup_enable	Enable wakeup timer
ertc_smooth_calibration_config	Configure smooth calibration
ertc_cal_output_select	Calibration output source selection
ertc_cal_output_enable	Enable calibration output
ertc_time_adjust	Time adjustment
ertc_daylight_set	Set daylight saving time
ertc_daylight_bpr_get	Get daylight saving time battery powered domain data register value (BPR)
ertc_refer_clock_detect_enable	Enable reference clock detection
ertc_direct_read_enable	Enable direct read mode
ertc_output_set	Set event output
ertc_timestamp_valid_edge_set	Set time stamp detection valid edge
ertc_timestamp_enable	Enable time stamp
ertc_timestamp_get	Get time stamp
ertc_timestamp_sub_second_get	Get time stamp subsecond
ertc_tamper_pull_up_enable	Enable tamper pin pull-up resistor
ertc_tamper_precharge_set	Set tamper pin precharge time
ertc_tamper_filter_set	Set tamper filter time
ertc_tamper_detect_freq_set	Set tamper detection frequency

ertc_tamper_valid_edge_set	Set tamper detection valid edge
ertc_tamper_timestamp_enable	Enable time stamp upon a tamper event
ertc_tamper_enable	Enable tamper detection
ertc_interrupt_enable	Enable interrupts
ertc_interrupt_get	Get the status of interrupt enable
ertc_flag_get	Get flag status
ertc_flag_clear	Clear flag
ertc_bpr_data_write	Write data to battery powered data register (BPR)
ertc_bpr_data_read	Read from battery powered data register (BPR)

5.6.1 ertc_num_to_bcd function

The table below describes the function ertc_num_to_bcd.

Table 105. 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);
```

5.6.2 ertc_bcd_to_num function

The table below describes the function ertc_bcd_to_num.

Table 106. 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);
```

5.6.3 ertc_write_protect_enable function

The table below describes the function ertc_write_protect_enable.

Table 107. 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();
```

5.6.4 ertc_write_protect_disable function

The table below describes the function ertc_write_protect_disable.

Table 108. 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();
```


5.6.5 ertc_wait_update function

The table below describes the function ertc_wait_update.

Table 109. 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();
```

5.6.6 ertc_wait_flag function

The table below describes the function ertc_wait_flag.

Table 110. 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 gets stuck here until flag status changes. This parameter can be SET or RESET.
Output parameter	NA
Return value	SUCCESS: flag state changed ERROR: flag wait timeout
Required preconditions	NA
Called functions	NA

flag

Flag selection

ERTC_ALAWF_FLAG: Alarm A write enable flag
 ERTC_WATWF_FLAG: Wakeup timer register write flag
 ERTC_TADJF_FLAG: Time adjustment flag
 ERTC_CALUPDF_FLAG: Calibration value update complete flag

Example:

```
ertc_wait_flag(ERTC_ALAWF_FLAG, RESET);
```

5.6.7 ertc_init_mode_enter function

The table below describes the function ertc_init_mode_enter.

Table 111. 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();
```

5.6.8 ertc_init_mode_exit function

The table below describes the function ertc_init_mode_exit.

Table 112. 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();
```

5.6.9 ertc_reset function

The table below describes the function ertc_reset.

Table 113. ertc_reset function

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

Example:

```
ertc_reset();
```

5.6.10 ertc_divider_set function

The table below describes the function ertc_divider_set.

Table 114. ertc_divider_set function

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

Example:

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

5.6.11 ertc_hour_mode_set function

The table below describes the function ertc_hour_mode_set.

Table 115. 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);
```

5.6.12 ertc_date_set function

The table below describes the function ertc_date_set.

Table 116. ertc_date_set function

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

Example:

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

5.6.13 ertc_time_set function

The table below describes the function ertc_time_set.

Table 117. ertc_time_set function

Name	Description
Function name	ertc_time_set
Function prototype	error_status ertc_time_set(uint8_t hour, uint8_t min, uint8_t sec, ertc_am_pm_type ampm);
Function description	Set time: hour, minute, second, AM/PM (for 12-hour format only)
Input parameter 1	hour: range 0~23
Input parameter 2	min: range 0~59
Input parameter3	sec: range 0~59
Input parameter4	ampm: AM/PM in 12-hour format (for 12-hour format only, don't care in 24-hour format) 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);
```

5.6.14 ertc_calendar_get function

The table below describes the function ertc_calendar_get.

Table 118. 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 “at32l021_ertc.h”:

typedef struct

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

} ertc_time_type;

year

Range 0~99

month

Range 1~12

day

Range 1~31

week

Range 1~7

hour

Range 0~23

min

Range 0~59

sec

Range 0~59

ampm

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

ERTC_AM: AM in 12 hour format

ERTC_PM: PM in 12 hour format

Example:

```
ertc_calendar_get(&time);
```

5.6.15 ertc_sub_second_get function

The table below describes the function ertc_sub_second_get.

Table 119. 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();
```

5.6.16 ertc_alarm_mask_set function

The table below describes the function ertc_alarm_mask_set.

Table 120. 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

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);
```

5.6.17 ertc_alarm_week_date_select function

The table below describes the function ertc_alarm_week_date_select.

Table 121. 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

wk

Alarm week/date format selection

ERTC_SELECT_DATE: Date mode

ERTC_SELECT_WEEK: Week mode

Example:

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


5.6.18 ertc_alarm_set function

The table below describes the function ertc_alarm_set.

Table 122. 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: range 1~31 Week: range 1~7
Input parameter3	hour: range 0~23
Input parameter4	min: range 0~59
Input parameter5	sec: range 0~59
Input parameter6	ampm: AM/PM in 12-hour format (12 hour format only, doesn't care in 24-hour format) Refer to the following description “ampm” for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

alarm_x

Alarm selection

ERTC_ALA: Alarm A

ampm

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

ERTC_24H: 24-hour format (for 24 hour format)

ERTC_AM: AM in 12-hour format

ERTC_PM: PM in 12-hour format

Example:

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

5.6.19 ertc_alarm_sub_second_set function

The table below describes the function ertc_alarm_sub_second_set.

Table 123. 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 parameter3	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

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);
```

5.6.20 ertc_alarm_enable function

The table below describes the function ertc_alarm_enable.

Table 124. 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

Example:

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

5.6.21 ertc_alarm_get function

The table below describes the function ertc_alarm_get.

Table 125. 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_alarm_value_type* alarm

ertc_alarm_value_type is defined in the “at32l021_ertc.h”:

```
typedef struct
{
```

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

day

Range 1~31

hour

Range 0~23

min

Range 0~59

sec

Range 0~59

ampm

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

ERTC_AM: AM in 12 hour format

ERTC_PM: PM in 12 hour format

mask

Alarm mask value, including:

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

week_date_sel

Alarm week/date format, including:

ERTC_SELECT_DATE: date mode

ERTC_SELECT_WEEK: week mode

week

Range 1~7

Example:

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

5.6.22 ertc_alarm_sub_second_get function

The table below describes the function ertc_alarm_sub_second_get.

Table 126. 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 value
Required preconditions	NA
Called functions	NA

alarm_x

Alarm selection

ERTC_ALA: Alarm A

Example:

```
ertc_alarm_sub_second_get(ERTC_ALA);
```

5.6.23 ertc_wakeup_clock_set function

The table below describes the function ertc_wakeup_clock_set.

Table 127. 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 clock source
Input parameter 1	Clock: wakeup clock source Refer to the following “clock” description for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

clock

Wakeup clock source selection:

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);
```

5.6.24 ertc_wakeup_counter_set function

The table below describes the function ertc_wakeup_counter_set.

Table 128. 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: wakeup counter value, ranging from 0 to 0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.6.25 ertc_wakeup_counter_get function

The table below describes the function ertc_wakeup_counter_get.

Table 129. 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 wakeup counter value
Input parameter 1	NA
Output parameter	NA
Return value	Wakeup counter value
Required preconditions	NA
Called functions	NA

Example:

```
ertc_wakeup_counter_get();
```

5.6.26 ertc_wakeup_enable function

The table below describes the function ertc_wakeup_enable

Table 130. 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 clock enable status This parameter can be TRUE or FALSE
Output parameter	NA
Return value	SUCCESS ERROR
Required preconditions	NA
Called functions	NA

Example:

```
ertc_wakeup_enable(TRUE);
```

5.6.27 ertc_smooth_calibration_config function

The table below describes the function ertc_smooth_calibration_config.

Table 131. 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 parameter3	clk_dec: reduce ERTC CLK cycles, range 0~511
Output parameter	NA
Return value	SUCCESS: Set success ERROR: Set error
Required preconditions	NA
Called functions	NA

period

Calibration periods

ERTC_SMOOTH_CAL_PERIOD_32: 32 seconds

ERTC_SMOOTH_CAL_PERIOD_16: 16 seconds

ERTC_SMOOTH_CAL_PERIOD_8: 8 seconds

clk_add

Add ERTC CLK

ERTC_SMOOTH_CAL_CLK_ADD_0: No effect

ERTC_SMOOTH_CAL_CLK_ADD_512: Add 512 ERTC_CLK cycles

Example:

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

5.6.28 ertc_cal_output_select function

The table below describes the function ertc_cal_output_select.

Table 132. 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);
```

5.6.29 ertc_cal_output_enable function

The table below describes the function ertc_cal_output_enable.

Table 133. 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);
```


5.6.30 ertc_time_adjust function

The table below describes the function ertc_time_adjust.

Table 134. 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: Set success ERROR: Set error
Required preconditions	NA
Called functions	NA

add1s

This bit is used to add seconds.

ERTC_TIME_ADD_NONE: No effect

ERTC_TIME_ADD_1S: Add 1 second

Example:

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

5.6.31 ertc_daylight_set function

The table below describes the function ertc_daylight_set.

Table 135. 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: Decrease 1 hour

save

Save daylight time

ERTC_DST_SAVE_0: set BPR bit to 0 in the CTRL register

ERTC_DST_SAVE_1: set BPR bit to 1 in the CTRL register

Example:

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

5.6.32 ertc_daylight_bpr_get function

The table below describes the function ertc_daylight_bpr_get.

Table 136. 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();
```

5.6.33 ertc_refer_clock_detect_enable function

The table below describes the function ertc_refer_clock_detect_enable.

Table 137. 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: Set success ERROR: Set error
Required preconditions	NA
Called functions	NA

Example:

```
ertc_refer_clock_detect_enable(TRUE);
```

5.6.34 ertc_direct_read_enable function

The table below describes the function ertc_direct_read_enable.

Table 138. 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);
```

5.6.35 ertc_output_set function

The table below describes the function ertc_output_set.

Table 139. 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 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 parameter3	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);
```

5.6.36 ertc_timestamp_valid_edge_set function

The table below describes the function ertc_timestamp_valid_edge_set.

Table 140. ertc_timestamp_valid_edge_set function

Name	Description
Function name	ertc_timestamp_valid_edge_set
Function prototype	void ertc_timestamp_valid_edge_set(ertc_timestamp_valid_edge_type edge);
Function description	Set timestamp detection valid edge
Input parameter 1	edge: timestamp detection valid edge Refer to the following “edge” descriptions for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

edge

Timestamp 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);
```

5.6.37 ertc_timestamp_enable function

The table below describes the function ertc_timestamp_enable.

Table 141. 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);
```

5.6.38 ertc_timestamp_get function

The table below describes the function ertc_timestamp_get.

Table 142. 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 “at32l021_ertc.h”:

typedef struct

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

} ertc_time_type;

year

Range 0~99

month

Range 1~12

day

Range 1~31

week

Range 1~7

hour

Range 0~23

min

Range 0~59

sec

Range 0~59

ampm

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

ERTC_AM: AM in 12-hour format

ERTC_PM: PM in 12-hour format

Example:

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

5.6.39 ertc_timestamp_sub_second_get function

The table below describes the function ertc_timestamp_sub_second_get.

Table 143. 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();
```

5.6.40 ertc_tamper_pull_up_enable function

The table below describes the function ertc_tamper_pull_up_enable.

Table 144. 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);
```

5.6.41 ertc_tamper_precharge_set function

The table below describes the function ertc_tamper_precharge_set.

Table 145. 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 function ertc_tamper_pull_up_enable.
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);
```

5.6.42 ertc_tamper_filter_set function

The table below describes the function ertc_tamper_filter_set.

Table 146. 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 sampling

ERTC_TAMPER_FILTER_4:

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

ERTC_TAMPER_FILTER_8:

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

Example:

```
ertc_tamper_filter_set(ERTC_TAMPER_FILTER_2);
```

5.6.43 ertc_tamper_detect_freq_set function

The table below describes the function ertc_tamper_detect_freq_set.

Table 147. 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);
```


5.6.44 ertc_tamper_valid_edge_set function

The table below describes the function ertc_tamper_valid_edge_set.

Table 148. 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 detection 1

trigger

Tamper detection valid edge selection

ERTC_TAMPER_EDGE_RISING: Rising edge

ERTC_TAMPER_EDGE_FALLING: Falling edge

ERTC_TAMPER_EDGE_LOW: Low level

ERTC_TAMPER_EDGE_HIGH: High level

Example:

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

5.6.45 ertc_tamper_timestamp_enable function

The table below describes the function ertc_tamper_timestamp_enable.

Table 149. ertc_tamper_timestamp_enable function

Name	Description
Function name	ertc_tamper_timestamp_enable
Function prototype	void ertc_tamper_timestamp_enable(confirm_state new_state);
Function description	Enable timestamp when a tamper event occurred
Input parameter 1	new_state: timestamp feature enable state when a tamper event occurred 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);
```

5.6.46 ertc_tamper_enable function

The table below describes the function ertc_tamper_enable.

Table 150. ertc_tamper_enable function

Name	Description
Function name	ertc_tamper_enable
Function prototype	void ertc_tamper_enable(ertc_tamper_select_type tamper_x, confirm_state new_state);
Function description	Enable tamper detection
Input parameter 1	tamper_x: tamper selection 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 detection 1

Example:

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

5.6.47 ertc_interrupt_enable function

The table below describes the function ertc_interrupt_enable.

Table 151. 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
Input parameter 1	source: interrupt source to be enabled Refer to the following “source” descriptions for details.
Input parameter 2	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_WAT_INT: Wakeup timer interrupt

ERTC_TS_INT: Time stamp interrupt

Example:

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

5.6.48 ertc_interrupt_get function

The table below describes the function ertc_interrupt_get.

Table 152. ertc_interrupt_get function

Name	Description
Function name	ertc_interrupt_get
Function prototype	flag_status ertc_interrupt_get(uint32_t source);
Function description	Get interrupt enable state
Input parameter 1	source: interrupt source 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_WAT_INT: Wakeup timer interrupt

ERTC_TS_INT: Time stamp interrupt

Example:

```
ertc_interrupt_get(ERTC_TP_INT);
```

5.6.49 ertc_flag_get function

The table below describes the function ertc_flag_get.

Table 153. 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_WATWF_FLAG: Wakeup timer register write flag

ERTC_TADJF_FLAG: Time adjust flag

ERTC_INITF_FLAG: Calendar initialization flag

ERTC_UPDF_FLAG:	Calendar update flag
ERTC_IMF_FLAG:	Initialization mode entry flag
ERTC_ALAF_FLAG:	Alarm A flag
ERTC_WATF_FLAG:	Wakeup timer flag
ERTC_TSF_FLAG:	Time stamp flag
ERTC_TSOF_FLAG:	Time stamp overflow flag
ERTC_TP1F_FLAG:	Tamper detection 1 flag
ERTC_CALUPDF_FLAG:	Calibration value update complete flag

Example:

```
ertc_flag_get(ERTC_TP1F_FLAG);
```

5.6.50 ertc_interrupt_flag_get function

The table below describes the function ertc_interrupt_flag_get.

Table 154. 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 interrupt flag status, and check the corresponding interrupt enable bit
Input parameter 1	flag: flag selection Refer to the following “flag” descriptions for details.
Output parameter	NA
Return value	Return SET or RESET
Required preconditions	NA
Called functions	NA

flag

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

ERTC_ALAF_FLAG:	Alarm A flag
ERTC_WATF_FLAG:	Wakeup timer flag
ERTC_TSF_FLAG:	Time stamp flag
ERTC_TP1F_FLAG:	Tamper detection 1 flag

Example:

```
ertc_interrupt_flag_get(ERTC_TP1F_FLAG);
```

5.6.51 ertc_flag_clear function

The table below describes the function ertc_flag_clear.

Table 155. 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_WATWF_FLAG:	Wakeup timer register write flag
ERTC_TADJF_FLAG:	Time adjust flag
ERTC_INITF_FLAG:	Calendar initialization flag
ERTC_UPDF_FLAG:	Calendar update flag
ERTC_IMF_FLAG:	Initialization mode entry flag
ERTC_ALAF_FLAG:	Alarm A flag
ERTC_WATF_FLAG:	Wakeup timer flag
ERTC_TSF_FLAG:	Time stamp flag
ERTC_TSOF_FLAG:	Time stamp overflow flag
ERTC_TP1F_FLAG:	Tamper detection 1 flag
ERTC_CALUPDF_FLAG:	Calibration value update complete flag

Example:

```
ertc_flag_clear(ERTC_TP1F_FLAG);
```

5.6.52 ertc_bpr_data_write function

The table below describes the function ertc_bpr_data_write.

Table 156. 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_DT3: Data register 3

ERTC_DT4: Data register 4

ERTC_DT5: Data register 5

Example:

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

5.6.53 ertc_bpr_data_read function

The table below describes the function ertc_bpr_data_read.

Table 157. 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	Data from BPR register
Required preconditions	NA
Called functions	NA

dt: Data register

ERTC_DT1: Data register 1

ERTC_DT2: Data register 2

ERTC_DT3: Data register 3

ERTC_DT4: Data register 4

ERTC_DT5: Data register 5

Example:

```
ertc_bpr_data_read(ERTC_DT1);
```

5.7 External interrupt/event controller (EXINT)

The EXINT register structure `exint_type` is defined in the “at32l021_exint.h”:

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

The table below gives a list of the EXINT registers:

Table 158. 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 159. 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	Read EXINT interrupt flag
exint_software_interrupt_event_generate	Software interrupt event generation
exint_interrupt_enable	Enable the selected EXINT interrupt
exint_event_enable	Enable the selected EXINT event

5.7.1 exint_reset function

The table below describes the function `exint_reset`.

Table 160. exint_reset function

Name	Description
Function name	<code>exint_reset</code>
Function prototype	<code>void exint_reset(void);</code>
Function description	Reset all EXINT registers to their reset values.
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	<code>crm_periph_reset();</code>

Example:

```
exint_reset ();
```

5.7.2 exint_default_para_init function

The table below describes the function `exint_default_para_init`.

Table 161. exint_default_para_init function

Name	Description
Function name	<code>exint_default_para_init</code>
Function prototype	<code>void exint_default_para_init(exint_init_type *exint_struct);</code>
Function description	Set the initial value for the EXINT initial structure
Input parameter 1	<code>exint_struct</code> : exint_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	The variable of <code>exint_init_type</code> must be defined before starting.
Called functions	NA

Example:

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


5.7.3 exint_init function

The table below describes the function `exint_init`.

Table 162. exint_init function

Name	Description
Function name	<code>exint_init</code>
Function prototype	<code>void exint_init(exint_init_type *exint_struct);</code>
Function description	Initialize EXINT
Input parameter 1	<i>exint_init_type</i> : <code>exint_init_struct</code> pointer
Output parameter	NA
Return value	NA
Required preconditions	The variable of <code>exint_init_type</code> must be set before starting.
Called functions	NA

The `exint_init_type` is defined in the "at32l021_exint.h":

`typedef struct`

```
{
    exint_line_mode_type          line_mode;
    uint32_t                      line_select;
    exint_polarity_config_type    line_polarity;
    confirm_state                 line_enable;
} exint_init_type;
```

line_mode

Select event mode or interrupt mode

EXINT_LINE_INTERRUPT: Interrupt mode

EXINT_LINE_EVENT: Event mode

line_select

Line selection

EXINT_LINE_NONE: No event

EXINT_LINE_0: line0

EXINT_LINE_1: line1

...

EXINT_LINE_25: line25

EXINT_LINE_26: line26

line_polarity

Trigger edge selection

EXINT_TRIGGER_RISING_EDGE: Rising edge

EXINT_TRIGGER_FALLING_EDGE: Falling edge

EXINT_TRIGGER_BOTH_EDGE: Rising/Falling edge

line_enable

Enable/disable line

FALSE: Disable line

TRUE: Enable line

Example:

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

```
exint_init_struct.line_enable = TRUE;  
exint_init_struct.line_mode = EXINT_LINE_INTERRUPT;  
exint_init_struct.line_select = EXINT_LINE_0;  
exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;  
exint_init(&exint_init_struct);
```

5.7.4 exint_flag_clear function

The table below describes the function `exint_flag_clear`.

Table 163. `exint_flag_clear` function

Name	Description
Function name	<code>exint_flag_clear</code>
Function prototype	<code>void exint_flag_clear(uint32_t exint_line);</code>
Function description	Clear the selected EXINT interrupt flag
Input parameter	<code>exint_line</code> : 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);
```

5.7.5 exint_flag_get function

The table below describes the function `exint_flag_get`.

Table 164. `exint_flag_get` function

Name	Description
Function name	<code>exint_flag_get</code>
Function prototype	<code>flag_status exint_flag_get(uint32_t exint_line);</code>
Function description	Get the selected EXINT interrupt flag
Input parameter	<code>exint_line</code> : line selection Refer to line_select for details.
Output parameter	NA
Return value	<code>flag_status</code> : 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);
```

5.7.6 exint_interrupt_flag_get function

The table below describes the function `exint_interrupt_flag_get`

Table 165. `exint_interrupt_flag_get` function

Name	Description
Function name	<code>exint_interrupt_flag_get</code>
Function prototype	<code>flag_status exint_interrupt_flag_get(uint32_t exint_line)</code>
Function description	Get EXINT interrupt flag status
Input parameter	<code>exint_line</code> : line selection Refer to line_select for details.
Output parameter	NA
Return value	<code>flag_status</code> : flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

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

5.7.7 exint_software_interrupt_event_generate function

The table below describes the function `exint_software_interrupt_event_generate`.

Table 166. `exint_software_interrupt_event_generate` function

Name	Description
Function name	<code>exint_software_interrupt_event_generate</code>
Function prototype	<code>void exint_software_interrupt_event_generate(uint32_t exint_line);</code>
Function description	Generate software interrupt event
Input parameter	<code>exint_line</code> : line selection Refer to 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);
```

5.7.8 exint_interrupt_enable function

The table below describes the function `exint_interrupt_enable`.

Table 167. exint_interrupt_enable function

Name	Description
Function name	<code>exint_interrupt_enable</code>
Function prototype	<code>void exint_interrupt_enable(uint32_t exint_line, confirm_state new_state);</code>
Function description	Enable the selected EXINT interrupt
Input parameter 1	<code>exint_line</code> : line selection Refer to line_select for details.
Input parameter 2	<code>new_state</code> : 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);
```

5.7.9 exint_event_enable function

The table below describes the function `exint_event_enable`.

Table 168. exint_event_enable function

Name	Description
Function name	<code>exint_event_enable</code>
Function prototype	<code>void exint_event_enable(uint32_t exint_line, confirm_state new_state);</code>
Function description	Enable the selected EXINT event
Input parameter 1	<code>exint_line</code> : line selection Refer to line_select for details.
Input parameter 2	<code>new_state</code> : 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);
```

5.8 Flash memory controller (FLASH)

The FLASH register structure flash_type is defined in the “at32l021_flash.h”:

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

The table below gives a list of the FLASH registers

Table 169. Summary of FLASH registers

Register	Description
flash_psr	Flash performance select register
flash_unlock	Flash unlock register
flash_usd_unlock	Flash user system data unlock register
flash_sts	Flash status register
flash_ctrl	Flash control register
flash_addr	Flash address register
flash_usd	User system data register
flash_epps	Erase/program protection status register
slib_sts0	Flash security library status register 0
slib_sts1	Flash security library status register 1
slib_pwd_clr	Flash security library password clear register
slib_misc_sts	Flash security library extra status register
Flash_crc_addr	Flash CRC address register
flash_crc_ctrl	Flash CRC check control register
flash_crc_chkr	Flash CRC check result register
slib_set_pwd	Flash security library password setting register
slib_set_range	Flash security library address setting register
em_slib_set	Extended memory security library setting register
btm_mode_set	Boot memory mode setting register
slib_unlock	Flash security library unlock register

The table below gives a list of FLASH library functions.

Table 170. Summary of FLASH library functions

Function name	Description
flash_flag_get	Get flag status
flash_flag_clear	Clear flag
flash_operation_wait_for	Wait for operation complete (Flash memory bank 1)
flash_operation_status_get	Get Flash operation status
flash_operation_wait_for	Wait for Flash operation complete
flash_unlock	Unlock Flash (Flash memory bank 1 and 2)
flash_lock	Lock Flash (Flash memory bank 1 and 2)
flash_sector_erase	Erase Flash sector
flash_internal_all_erase	Erase internal Flash
flash_user_system_data_erase	Erase user system data
flash_word_program	Flash word programming
flash_halfword_program	Flash half-word programming
flash_byte_program	Flash byte programming
flash_user_system_data_program	User system data programming
flash_epp_set	Erase/programming protection configuration
flash_epp_status_get	Get erase/programming protection status
flash_fap_enable	Flash low level access protection enable
flash_fap_status_get	Get Flash low level access protection status
flash_fap_high_level_enable	Flash high level access protection enable
flash_fap_high_level_status_get	Get Flash high level access protection status
flash_ssb_set	System configuration byte configuration
flash_ssb_status_get	Get system configuration byte configuration status
flash_interrupt_enable	Flash interrupt configuration
flash_slib_enable	sLib enable
flash_slib_disable	sLib disable
flash_slib_state_get	Get sLib states
flash_slib_start_sector_get	Get sLib start sector
flash_slib_datastart_sector_get	Get sLib data area start sector
flash_slib_end_sector_get	Get sLib end sector
flash_crc_calibrate	Flash CRC verify
flash_boot_memory_extension_mode_enable	Boot memory is used as an extended Flash memory
flash_extension_memory_slib_enable	Extended Flash memory is used as a security library
flash_extension_memory_slib_state_get	Get the status of extended Flash memory which is used as a security library
flash_em_slib_inststart_sector_get	Get the start page of instruction area of security library in the extended memory

5.8.1 flash_flag_get function

The table below describes the function flash_flag_get.

Table 171. flash_flag_get function

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

flash_flag

Flag selection.

FLASH_OBF_FLAG:	Flash operation busy
FLASH_ODF_FLAG:	Flash operation complete
FLASH_PRGMERR_FLAG:	Flash programming error
FLASH_EPPERR_FLAG:	Flash erase error
FLASH_USDERR_FLAG:	User system data area error

Example:

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

5.8.2 flash_flag_clear function

The table below describes the function flash_flag_clear.

Table 172. 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

Example:

```
flash_flag_clear(FLASH_ODF_FLAG);
```

5.8.3 flash_operation_status_get function

The table below describes the function flash_operation_status_get.

Table 173. flash_operation_status_get function

Name	Description
Function name	flash_operation_status_get
Function prototype	flash_status_type flash_operation_status_get(void);
Function description	Get operation status
Input parameter	NA
Output parameter	NA
Return value	Refer to " flash_status_type " below for details.
Required preconditions	NA
Called functions	NA

flash_status_type

FLASH_OPERATE_BUSY Operate busy

FLASH_PROGRAM_ERROR Programming error

FLASH_EPP_ERROR Erase/program protection error

FLASH_OPERATE_DONE Operation complete

FLASH_OPERATE_TIMEOUT: Flash operation timeout

Example:

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


5.8.4 flash_operation_wait_for function

The table below describes the function flash_operation_wait_for.

Table 174. 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.
Output parameter	NA
Return value	Refer to "flash_status_type" in 5.8.3 flash_operation_status_get function 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);
```

5.8.5 flash_unlock function

The table below describes the function flash_unlock.

Table 175. flash_unlock function

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

Example:

```
flash_unlock();
```

5.8.6 flash_lock function

The table below describes the function flash_lock.

Table 176. flash_lock function

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

Example:

```
flash_lock();
```

5.8.7 flash_sector_erase function

The table below describes the function flash_sector_erase.

Table 177. flash_sector_erase function

Name	Description
Function name	flash_sector_erase
Function prototype	flash_status_type flash_sector_erase(uint32_t sector_address);
Function description	Erase data in the selected Flash sector address
Input parameter	sector_address: select the Flash sector address to be erased, usually Flash sector start address
Output parameter	NA
Return value	Refer to "flash_status_type" in 5.8.3 flash_operation_status_get function for details.
Required preconditions	NA
Called functions	NA

Example:

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

5.8.8 flash_internal_all_erase function

The table below describes the function flash_internal_all_erase.

Table 178. flash_internal_all_erase function

Name	Description
Function name	flash_internal_all_erase
Function prototype	flash_status_type flash_internal_all_erase(void);
Function description	Erase internal Flash data
Input parameter	NA
Output parameter	NA
Return value	Refer to "flash_status_type" in 5.8.3 flash_operation_status_get function for details.
Required preconditions	NA
Called functions	NA

Example:

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

5.8.9 flash_user_system_data_erase function

The table below describes the function flash_user_system_data_erase.

Table 179. flash_user_system_data_erase function

Name	Description
Function name	flash_user_system_data_erase
Function prototype	flash_status_type flash_user_system_data_erase(void);
Function description	Erase user system data
Input parameter	NA
Output parameter	NA
Return value	Refer to "flash_status_type" in 5.8.3 flash_operation_status_get function 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();
```

5.8.10 flash_word_program function

The table below describes the function flash_word_program.

Table 180. 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	Refer to "flash_status_type" in 5.8.3 flash_operation_status_get function 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(l = 0; l < 256; i++)
    {
        status = flash_word_program(0x08001000 + i*4, i);
    }
}
```

5.8.11 flash_halfword_program function

The table below describes the function flash_halfword_program.

Table 181. flash_halfword_program function

Name	Description
Function name	flash_halfword_program
Function prototype	flash_status_type flash_halfword_program(uint32_t address, uint16_t data);
Function description	Write a half-word data to a given address
Input parameter 1	Address: programmed address, half-word-aligned
Input parameter 2	Data: programmed data
Output parameter	NA
Return value	Refer to "flash_status_type" in 5.8.3 flash_operation_status_get function 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);
    }
}
```

5.8.12 flash_byte_program function

The table below describes the function flash_byte_program.

Table 182. flash_byte_program function

Name	Description
Function name	flash_byte_program
Function prototype	flash_status_type flash_byte_program(uint32_t address, uint8_t data);
Function description	Program a byte data to a given address
Input parameter 1	Address: programmed address
Input parameter 2	Data: programmed data
Output parameter	NA
Return value	Refer to "flash_status_type" in 5.8.3 flash_operation_status_get function 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);
    }
}
```

5.8.13 flash_user_system_data_program function

The table below describes the function flash_user_system_data_program.

Table 183. flash_user_system_data_program function

Name	Description
Function name	flash_user_system_data_program
Function prototype	flash_status_type flash_user_system_data_program (uint32_t address, uint8_t data);
Function description	Program a byte data to a given address in the user system data area
Input parameter 1	Address: programmed address
Input parameter 2	Data: programmed data
Output parameter	NA
Return value	Refer to "flash_status_type" in 5.8.3 flash_operation_status_get function 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);
}
```

5.8.14 flash_epp_set function

The table below describes the function flash_epp_set.

Table 184. flash_epp_set function

Name	Description
Function name	flash_epp_set
Function prototype	flash_status_type flash_epp_set(uint32_t *sector_bits);
Function description	Enable erase programming protection
Input parameter	*sector_bits: Erase programming protection sector address pointer. Each bit in bits 15~0 protects 4KB sectors, bit 31 protects Flash extension area. Setting this bit to 1 enables sector protection.
Output parameter	NA
Return value	Refer to "flash_status_type" in 5.8.3 flash_operation_status_get function 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);
}
```

5.8.15 flash_epp_status_get function

The table below describes the function flash_epp_status_get.

Table 185. 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 15~0 protects 4KB sectors, bit 31 protects Flash extension area. Setting this bit to 1 enables sector protection.
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.8.16 flash_fap_enable function

The table below describes the function flash_fap_enable.

Table 186. flash_fap_enable function

Name	Description
Function name	flash_fap_enable
Function prototype	flash_status_type flash_fap_enable(confirm_state new_state);
Function description	Enable Flash access protection
Input parameter	new_state: Flash access protection status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	Refer to "flash_status_type" in 5.8.3 flash_operation_status_get function 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);
```

5.8.17 flash_fap_status_get function

The table below describes the function flash_fap_status_get.

Table 187. flash_fap_status_get function

Name	Description
Function name	flash_fap_status_get
Function prototype	flag_status flash_fap_status_get(void);
Function description	Get the status of Flash access protection
Input parameter	NA
Output parameter	NA
Return value	flag_status: flag status This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

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

5.8.18 flash_fap_high_level_enable

The table below describes the function flash_fap_high_level_enable.

Table 188. flash_fap_high_level_enable function

Name	Description
Function name	flash_fap_high_level_enable
Function prototype	flash_status_type flash_fap_high_level_enable(void);
Function description	Enable Flash high level access protection
Input parameter	NA
Output parameter	NA
Return value	flag_status: flag status Refer to "flash_status_type" in 5.8.3 flash_operation_status_get function for details.
Required preconditions	NA
Called functions	NA

Note: Once Flash high level access protection is enabled, SWD interface will permanently be disabled.

Example:

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

5.8.19 flash_fap_high_level_status_get

The table below describes the function flash_fap_high_level_status_get.

Table 189. flash_fap_high_level_status_get function

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

Example:

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

5.8.20 flash_ssb_set function

The table below describes the function flash_ssb_set.

Table 190. flash_ssb_set function

Name	Description
Function name	flash_ssb_set
Function prototype	flash_status_type flash_ssb_set(uint8_t usd_ssb);
Function description	Configure system setting bytes
Input parameter	usd_ssb: system setting byte value is a combination of the selected data from all data group, refer to <i>ssb_data_define</i> for details.
Output parameter	NA
Return value	Refer to "flash_status_type" in 5.8.3 flash_operation_status_get function for details.
Required preconditions	NA
Called functions	NA

ssb_data_define

type 1:

USD_WDT_ATO_DISABLE: Watchdog auto-start disabled

USD_WDT_ATO_ENABLE: Watchdog auto-start enabled

type 2:

USD_DEPSLP_NO_RST: No reset occurred when entering Deepsleep mode

USD_DEPSLP_RST: Reset occurred when entering Deepsleep mode

type 3:

USD_STDBY_NO_RST: No reset occurred when entering Standby mode

USD_STDBY_RST: Reset occurred when entering Standby mode

type 4:

USD_BOOT1_LOW BOOT1 LOW

USD_BOOT1_HIGH BOOT1 HIGH

USD_WDT_DEPSLP_CONTINUE: WDT continues counting in Deepsleep mode

USD_WDT_DEPSLP_STOP: WDT stops counting in Deepsleep mode

USD_WDT_STDBY_CONTINUE: WDT continues counting in Standby mode

USD_WDT_STDBY_STOP: WDT stops counting in Standby mode

Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
status = flash_user_system_data_erase();
if(status == FLASH_OPERATE_DONE)
{
    status = flash_ssb_set(USD_WDT_ATO_DISABLE | USD_DEPSLP_NO_RST | USD_STDBY_RST |
FLASH_BOOT_FROM_BANK1);
}
```

5.8.21 flash_ssb_status_get function

The table below describes the function flash_ssb_status_get.

Table 191. flash_ssb_status_get function

Name	Description
Function name	flash_ssb_status_get
Function prototype	uint8_t flash_ssb_status_get(void);
Function description	Get the status of system setting bytes
Input parameter	NA
Output parameter	NA
Return value	Return system setting byte value, refer to ssb_data_define for details.
Required preconditions	NA
Called functions	NA

Example:

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

5.8.22 flash_interrupt_enable function

The table below describes the function flash_interrupt_enable.

Table 192. 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 below 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

Example:

```
flash_interrupt_enable(FLASH_ERR_INT | FLASH_ODF_INT, TRUE);
```

5.8.23 flash_slrib_enable function

The table below describes the function flash_slrib_enable.

Table 193. flash_slrib_enable function

Name	Description
Function name	flash_slrib_enable
Function prototype	flash_status_type flash_slrib_enable(uint32_t pwd, uint16_t start_sector, uint16_t inst_start_sector, uint16_t end_sector);
Function description	Enable security library (sLib) and its address range
Input parameter 1	Pwd: sLib password. The sLib data are saved as ciphertext, associated with encrypted computing. A correct password is entered in order to unlock encryption.
Input parameter 2	start_sector: sLib start sector number
Input parameter 3	inst_start_sector: sLib data area instruction start sector number
Input parameter 4	end_sector: sLib end sector number
Output parameter	NA
Return value	Refer to "flash_status_type" in 5.8.3 flash_operation_status_get function for details.
Required preconditions	NA
Called functions	NA

Example:

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

5.8.24 flash_slib_disable function

The table below describes the function flash_slib_disable.

Table 194. 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 ERROE or SUCCESS.
Required preconditions	NA
Called functions	NA

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

Example:

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

5.8.25 flash_slib_state_get function

The table below describes the function flash_slib_state_get.

Table 195. 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();
```

5.8.26 flash_slb_start_sector_get function

The table below describes the function flash_slb_start_sector_get.

Table 196. flash_slb_start_sector_get function

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

Example:

```
uint16_t num;  
num = flash_slb_start_sector_get();
```

5.8.27 flash_slb_inststart_sector_get function

The table below describes the function flash_slb_inststart_sector_get.

Table 197. flash_slb_inststart_sector_get function

Name	Description
Function name	flash_slb_inststart_sector_get
Function prototype	uint16_t flash_slb_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_slb_inststart_sector_get();
```

5.8.28 flash_slib_end_sector_get function

The table below describes the function flash_slib_end_sector_get.

Table 198. 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();
```

5.8.29 flash_crc_calibrate function

The table below describes the function flash_crc_calibrate.

Table 199. flash_crc_calibrate function

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


5.8.30 flash_boot_memory_extension_mode_enable

The table describes the flash_boot_memory_extension_mode_enable

Table 200. flash_boot_memory_extension_mode_enable

Name	Description
Function name	flash_boot_memory_extension_mode_enable
Function prototype	void flash_boot_memory_extension_mode_enable (void);
Function description	Boot memory is used as extended Flash memory
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Note: This feature is irreversible, so it can only be enabled once. After enabled, it is impossible to restore its function as boot memory.

Example:

```
flash_boot_memory_extension_mode_enable ();
```

5.8.31 flash_extension_memory_slib_enable

The table describes the flash_extension_memory_slib_enable

Table 201. flash_extension_memory_slib_enable

Name	Description
Function name	flash_extension_memory_slib_enable
Function prototype	void flash_extension_memory_slib_enable (uint32_t pwd, uint16_t inst_start_sector);
Function description	Extended Flash memory is used as security library, and the sLib data area start sector is set
Input parameter 1	Pwd: sLib password, data in the sLib area is encrypted, so a correct password is needed to unlock sLib.
Input parameter 2	inst_start_sector: the start sector of security library instruction area
Output parameter	NA
Return value	Refer to "flash_status_type" in 5.8.3 flash_operation_status_get function for details.
Required preconditions	NA
Called functions	NA

Note: Either Flash memory or its extended area can be set as sLib. They cannot be set as sLib at the same time.

Example:

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

5.8.32 flash_extension_memory_slib_state_get

The table describes the flash_extension_memory_slib_state_get

Table 202. flash_extension_memory_slib_state_get

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

Example:

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

5.8.33 flash_em_slib_inststart_sector_get

The table describes the flash_em_slib_inststart_sector_get

Table 203. flash_em_slib_inststart_sector_get

Name	Description
Function name	flash_em_slib_inststart_sector_get
Function prototype	uint16_t flash_em_slib_inststart_sector_get (void);
Function description	Get the start sector number of security library instruction area (when Flash extension memory is set as sLib)
Input parameter	NA
Output parameter	NA
Return value	Return the start sector number of security library instruction area when the extended Flash memory is set as a security library
Required preconditions	NA
Called functions	NA

Example:

```
uint16_t num;
num = flash_em_slib_inststart_sector_get();
```

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

The GPIO register structure `gpio_type` is defined in the “at32l021_gpio.h”:

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

} gpio_type;
```

The table below gives a list of the GPIO registers

Table 204. 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 register
odt	GPIO output register
scr	GPIO set/clear register
wpr	GPIO write protection register
muxl	GPIO multiplexed function low register
muxh	GPIO multiplexed function high register
clr	GPIO port bit clear register
togr	GPIO bit toggle register
hdrv	GPIO huge current control register

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

Table 205. 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 pins
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

5.9.1 gpio_reset function

The table below describes the function gpio_reset.

Table 206. gpio_reset function

Name	Description
Function name	gpio_reset
Function prototype	void gpio_reset(gpio_type *gpio_x);
Function description	GPIO is reset by CRM reset register
Input parameter	gpio_x: Select a GPIO peripheral. GPIOA, GPIOB, GPIOC, GPIOF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

Example:

```
gpio_reset(GPIOA);
```

5.9.2 gpio_init function

The table below describes the function gpio_init.

Table 207. gpio_init function

Name	Description
Function name	gpio_init
Function prototype	void gpio_init(gpio_type *gpio_x, gpio_init_type *gpio_init_struct);
Function description	Initialize GPIO peripherals
Input parameter 1	gpio_x: the selected GPIO peripheral GPIOA, GPIOB, GPIOC, GPIOF
Input parameter 2	gpio_init_struct: gpio_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

gpio_init_type structure

The gpio_init_type is defined in the at32l021_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);
  
```

5.9.3 gpio_default_para_init function

The table below describes the function gpio_default_para_init.

Table 208. gpio_default_para_init function

Name	Description
Function name	gpio_default_para_init
Function prototype	void gpio_default_para_init(gpio_init_type *gpio_init_struct);
Function description	Initialize GPIO default parameters
Input parameter	gpio_init_struct: gpio_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

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

Table 209. 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);
```

5.9.4 gpio_input_data_bit_read function

The table below describes the function gpio_input_data_bit_read.

Table 210. 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: indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC or GPIOF.
Input parameter 2	Pins: indicates the GPIO pins, refer to <i>gpio_pins</i> 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);
```

5.9.5 gpio_input_data_read function

The table below describes the function gpio_input_data_read.

Table 211. gpio_input_data_read function

Name	Description
Function name	gpio_input_data_read
Function prototype	uint16_t gpio_input_data_read(gpio_type *gpio_x);
Function description	Read GPIO input ports
Input parameter	gpio_x: indicates the selected GPIO peripheral. This parameter can be GPIOA, GPIOB, GPIOC, GPIOF
Output parameter	NA
Return value	Return GPIO input port status
Required preconditions	NA
Called functions	NA

Example:

```
gpio_input_data_read(GPIOA);
```

5.9.6 gpio_output_data_bit_read function

The table below describes the function gpio_output_data_bit_read.

Table 212. gpio_output_data_bit_read function

Name	Description
Function name	gpio_output_data_bit_read
Function prototype	uint16_t gpio_output_data_bit_read(gpio_type *gpio_x);
Function description	Read GPIO output port pin
Input parameter 1	gpio_x: indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC or GPIOF.
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);
```

5.9.7 gpio_output_data_read function

The table below describes the function `gpio_output_data_read`.

Table 213. gpio_output_data_read function

Name	Description
Function name	<code>gpio_output_data_read</code>
Function prototype	<code>uint16_t gpio_output_data_read(gpio_type *gpio_x);</code>
Function description	Read GPIO output port
Input parameter	<code>gpio_x</code> : the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOF
Output parameter	NA
Return value	Read GPIO output port status
Required preconditions	NA
Called functions	NA

Example:

```
gpio_output_data_read(GPIOA);
```

5.9.8 gpio_bits_set function

The table below describes the function `gpio_bits_set`.

Table 214. gpio_bits_set function

Name	Description
Function name	<code>gpio_bits_set</code>
Function prototype	<code>void gpio_bits_set(gpio_type *gpio_x, uint16_t pins);</code>
Function description	Set GPIO pins
Input parameter 1	<code>gpio_x</code> : indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC or GPIOF
Input parameter 2	<code>Pins</code> : 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);
```


5.9.9 gpio_bits_reset function

The table below describes the function `gpio_bits_reset`.

Table 215. gpio_bits_reset function

Name	Description
Function name	<code>gpio_bits_reset</code>
Function prototype	<code>void gpio_bits_reset(gpio_type *gpio_x, uint16_t pins);</code>
Function description	Reset GPIO pins
Input parameter 1	<code>gpio_x</code> : indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOF
Input parameter 2	<code>Pins</code> : 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);
```

5.9.10 gpio_bits_toggle function

The table below describes the function `gpio_bits_toggle`.

Table 216. gpio_bits_toggle function

Name	Description
Function name	<code>gpio_bits_toggle</code>
Function prototype	<code>void gpio_bits_toggle(gpio_type *gpio_x, uint16_t pins);</code>
Function description	Reset GPIO pins
Input parameter 1	<code>gpio_x</code> : indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOF
Input parameter 2	<code>Pins</code> : 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);
```

5.9.11 gpio_bits_write function

The table below describes the function gpio_bits_write.

Table 217. gpio_bits_write function

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

Example:

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

5.9.12 gpio_port_write function

The table below describes the function gpio_port_write.

Table 218. gpio_port_write function

Name	Description
Function name	gpio_port_write
Function prototype	void gpio_port_write(gpio_type *gpio_x, uint16_t port_value);
Function description	Write GPIO ports
Input parameter 1	gpio_x: indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOF
Input parameter 2	port_value: indicates the port value to write This parameter can be 0x0000~0xFFFF.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.9.13 gpio_pin_wp_config function

The table below describes the function `gpio_pin_wp_config`.

Table 219. gpio_pin_wp_config function

Name	Description
Function name	<code>gpio_pin_wp_config</code>
Function prototype	<code>void gpio_pin_wp_config(gpio_type *gpio_x, uint16_t pins);</code>
Function description	Configure GPIO pin write protection
Input parameter 1	gpio_x: indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOF
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);
```

5.9.14 gpio_pins_huge_driven_config function

The table below describes the function `gpio_pins_huge_driven_config`.

Table 220. gpio_pins_huge_driven_config function

Name	Description
Function name	<code>gpio_pins_huge_driven_config</code>
Function prototype	<code>void gpio_pins_huge_driven_config(gpio_type *gpio_x, uint16_t pins, confirm_state new_state);</code>
Function description	Configure huge drive capability of GPIO pins
Input parameter 1	gpio_x: indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOF
Input parameter 2	Pins: indicates the GPIO pins, refer to gpio_pins for details.
Input parameter 3	new_state: the status of to-be-configured huge current sourcing/sinking capability 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);
```

5.9.15 gpio_pin_mux_config function

The table below describes the function gpio_pin_mux_config.

Table 221. gpio_pin_mux_config function

Name	Description
Function name	gpio_pin_mux_config
Function prototype	void gpio_pin_mux_config(gpio_type *gpio_x, gpio_pins_source_type gpio_pin_source, gpio_mux_sel_type gpio_mux);
Function description	Configure GPIO pin multiplexed function
Input parameter 1	gpio_x: the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOF
Input parameter 2	gpio_pin_source: GPIO pin to be configured
Input parameter 3	gpio_mux: IOMUX index to be configured
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

gpio_pin_source

Set GPIO pins

GPIO_PINS_SOURCE0:	GPIO pin 0
GPIO_PINS_SOURCE1:	GPIO pin 1
GPIO_PINS_SOURCE2:	GPIO pin 2
GPIO_PINS_SOURCE3:	GPIO pin 3
GPIO_PINS_SOURCE4:	GPIO pin 4
GPIO_PINS_SOURCE5:	GPIO pin 5
GPIO_PINS_SOURCE6:	GPIO pin 6
GPIO_PINS_SOURCE7:	GPIO pin 7
GPIO_PINS_SOURCE8:	GPIO pin 8
GPIO_PINS_SOURCE9:	GPIO pin 9
GPIO_PINS_SOURCE10:	GPIO pin 10
GPIO_PINS_SOURCE11:	GPIO pin 11
GPIO_PINS_SOURCE12:	GPIO pin 12
GPIO_PINS_SOURCE13:	GPIO pin 13
GPIO_PINS_SOURCE14:	GPIO pin 14
GPIO_PINS_SOURCE15:	GPIO pin 15

gpio_mux: Select IOMUX index

GPIO_MUX_0
GPIO_MUX_1
GPIO_MUX_2
GPIO_MUX_3
GPIO_MUX_4
GPIO_MUX_5
GPIO_MUX_6
GPIO_MUX_7
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);
```

5.10 Hardware integer divider (HWDIV)

HWDIV register structure `hwdiv_type` is defined in the “at32l021_hwdiv.h”:

```
/**
 * @brief type define hwdiv register all
 */
typedef struct
{
    ...
} hwdiv_type;
```

The table below gives a list of the HWDIV registers.

Table 222. Summary of HWDIV registers

Register	Description
<code>dvdd</code>	HWDIV dividend register
<code>dvsr</code>	HWDIV divider register
<code>quot</code>	HWDIV quotient register
<code>remd</code>	HWDIV remainder register
<code>ctrl</code>	HWDIV control register
<code>sts</code>	HWDIV status register

The table below gives a list of HWDIV library functions.

Table 223. Summary of HWDIV library functions

Function name	Description
<code>hwdiv_dividend_set</code>	Set a 32-bit dividend
<code>hwdiv_divisor_set</code>	Set a 32-bit divisor, and calculate quotient and remainder
<code>hwdiv_quotient_get</code>	Get quotient
<code>hwdiv_remainder_get</code>	Get remainder
<code>hwdiv_interrupt_enale</code>	Enable or disable error interrupts
<code>hwdiv_signed_operate_enable</code>	Enable signed integer operation
<code>hwdiv_flag_get</code>	Get flag status
<code>hwdiv_flag_clear</code>	Clear flag status

5.10.1 hwdiv_dividend_set function

The table below describes the function hwdiv_dividend_set.

Table 224. hwdiv_dividend_set function

Name	Description
Function name	hwdiv_dividend_set
Function prototype	void hwdiv_dividend_set(uint32_t value);
Function description	Set dividend
Input parameter 1	value: 32-bit 的整数
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set dividend */  
hwdiv_dividend_set(0x98765432);
```

5.10.2 hwdiv_divisor_set function

The table below describes the function hwdiv_divisor_set.

Table 225. hwdiv_divisor_set function

Name	Description
Function name	hwdiv_divisor_set
Function prototype	void hwdiv_divisor_set(uint32_t value);
Function description	Set divisor and enable operation
Input parameter 1	value: 32-bit integer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* set divisor */  
hwdiv_divisor_set(0x12345678);
```

5.10.3 hwdiv_quotient_set function

The table below describes the function hwdiv_quotient_get.

Table 226. hwdiv_quotient_get function

Name	Description
Function name	hwdiv_quotient_get
Function prototype	uint32_t hwdiv_quotient_get(void);
Function description	Get quotient
Input parameter	NA
Output parameter	NA
Return value	uint32_t: 32-bit integer quotient
Required preconditions	NA
Called functions	NA

Example:

```
/* get quotient */  
uint32 quot;  
quot = hwdiv_quotient_get();
```

5.10.4 hwdiv_remainder_get function

The table below describes the function hwdiv_remainder_get.

Table 227. hwdiv_remainder_get function

Name	Description
Function name	hwdiv_remainder_get
Function prototype	uint32_t hwdiv_remainder_get(void);
Function description	Get remainder
Input parameter	NA
Output parameter	NA
Return value	uint32_t: 32-bit integer remainder
Required preconditions	NA
Called functions	NA

Example:

```
/* get remainder */  
uint32 remd;  
remd = hwdiv_remainder_get();
```


5.10.5 hwdiv_interrupt_enale function

The table below describes the function hwdiv_interrupt_enale.

Table 228. hwdiv_interrupt_enale function

Name	Description
Function name	hwdiv_interrupt_enale
Function prototype	void hwdiv_interrupt_enale(confirm_state new_state);
Function description	Enable or disable error interrupts
Input parameter	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 interrupt */
hwdiv_interrupt_enale(true);
```

5.10.6 hwdiv_signed_operate_enablefunction

The table below describes the function hwdiv_signed_operate_enable.

Table 229. hwdiv_signed_operate_enable function

Name	Description
Function name	hwdiv_signed_operate_enable
Function prototype	void hwdiv_signed_operate_enable(confirm_state new_state);
Function description	Enable or disable signed integer operation
Input parameter	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 signed operation */
hwdiv_signed_operate_enable(true);
```

5.10.7 hwddiv_flag_get_enable function

The table below describes the function hwddiv_flag_get.

Table 230. hwddiv_flag_get function

Name	Description
Function name	hwddiv_flag_get
Function prototype	flag_status hwddiv_flag_get(uint32_t flag);
Function description	Get flag status
Input parameter	Flag: select a flag
Output parameter	NA
Return value	flag_status: flag status This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

```
/* get flag status */
while(hwddiv_flag_get(HWDIV_DIV0F_FLAG) == SET);
```

5.10.8 hwddiv_interrupt_flag_get function

The table below describes the function hwddiv_interrupt_flag_get

Table 231. hwddiv_interrupt_flag_get function

Name	Description
Function name	hwddiv_interrupt_flag_get
Function prototype	flag_status hwddiv_interrupt_flag_get(uint32_t flag);
Function description	Get interrupt flag status
Input parameter	Flag: select a flag
Output parameter	NA
Return value	flag_status: flag status This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

```
/* get interrupt flag status */
while(hwddiv_interrupt_flag_get(HWDIV_DIV0F_FLAG) == SET);
```

5.10.9 hwdiv_flag_clear function

The table below describes the function hwdiv_flag_clear

Table 232. hwdiv_flag_clear function

Name	Description
Function name	hwdiv_flag_clear
Function prototype	void hwdiv_flag_clear(uint32_t flag);
Function description	Clear flag
Input parameter	Flag: select a flag
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* clear flag status */  
hwdiv_error_clear(HWDIV_DIV0F_FLAG);
```

5.11 I2C interfaces

The I2C register structure `i2c_type` is defined in the “at32l021_i2c.h”:

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

} i2c_type;
```

The table below gives a list of the I2C registers

Table 233. Summary of I2C register

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	I2C Clock control register
timeout	I2C timeout register
sts	I2C Status register
clr	I2C Status clear register
pec	PEC register
rxdt	I2C receive register
txdt	I2C transmit register

The table below gives a list of I2C library functions.

Table 234. Summary of I2C library functions

Function name	Description
i2c_reset	I2C peripheral reset
i2c_init	I2C initialization, set bus speed, 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 peripheral
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 slave address sent by master

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_smbus_alert_get	Get SMBus alert pin level
i2c_slave_data_ctrl_enable	Enable slave single-byte receive control
i2c_pec_calculate_enable	Enable PEC calculation
i2c_pec_transmit_enable	Enable PEC transmit
i2c_pec_value_get	Get current PEC value
i2c_timeout_set	Set clock level timeout detection
i2c_timeout_detect_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
i2c_wakeup_enable	Enable wakeup from DeepSleep mode
i2c_analog_filter_enable	Enable analog filter

Table 235. 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)

Function name	Description
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 sends data (polling mode)
i2c_smbus_master_receive	SMBus master receives data (polling mode)
i2c_smbus_slave_transmit	SMBus slave sends data (polling mode)
i2c_smbus_slave_receive	SMBus slave receives data (polling mode)
i2c_memory_write	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

5.11.1 i2c_reset function

The table below describes the function i2c_reset.

Table 236. i2c_reset function

Name	Description
Function name	i2c_reset
Function prototype	void i2c_reset(i2c_type *i2c_x)
Function description	Reset all I2C registers to their initial values through CRM (Clock and reset management)
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.2 i2c_init function

The table below describes the function i2c_init.

Table 237. 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: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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 the AN0091_AT32F402_405_I2C_Application_Note.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

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

5.11.3 i2c_own_address1_set function

The table below describes the function i2c_own_address1_set.

Table 238. i2c_own_address1_set function

Name	Description
Function name	i2c_own_address1_set
Function prototype	void i2c_own_address1_set(i2c_type *i2c_x, i2c_address_mode_type mode, uint16_t address);
Function description	Set own address 1
Input parameter 1	Mode: Own address 1 address mode Refer to the "mode" description below for details.
Input parameter 2	Address: own address 1
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

mode

Own address 1 address mode

I2C_ADDRESS_MODE_7BIT: 7-bit address

I2C_ADDRESS_MODE_10BIT: 10-bit address

Example:

```
i2c_own_address1_set(I2C1, I2C_ADDRESS_MODE_7BIT, 0xA0);
```

5.11.4 i2c_own_address2_set function

The table below describes the function i2c_own_address2_set.

Table 239. 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: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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] only
I2C_ADDR2_MASK07:	All non-I2C reserved addresses would respond

Example:

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


5.11.5 i2c_own_address2_enable function

The table below describes the function i2c_own_address2_enable.

Table 240. i2c_own_address2_enable function

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

Example:

```
i2c_own_address2_enable(I2C1, TRUE);
```

5.11.6 i2c_smbus_enable function

The table below describes the function i2c_smbus_enable.

Table 241. 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: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
Input parameter 2	Mode: SMBus mode selection Refer to the following “mode” descriptions for details.
Input parameter 3	new_state: indicates SMBus mode status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

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);
```

5.11.7 i2c_enable function

The table below describes the function i2c_enable.

Table 242. i2c_enable function

Name	Description
Function name	i2c_enable
Function prototype	void i2c_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable I2C peripheral
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.8 i2c_clock_stretch_enable function

The table below describes the function i2c_clock_stretch_enable.

Table 243. i2c_clock_stretch_enable function

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

5.11.9 i2c_ack_enable function

The table below describes the function i2c_ack_enable.

Table 244. 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	Enable ACK and NACK. 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: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.10 i2c_addr10_mode_enable function

The table below describes the function i2c_addr10_mode_enable.

Table 245. 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: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.11 i2c_transfer_addr_set function

The table below describes the function i2c_transfer_addr_set.

Table 246. i2c_transfer_addr_set function

Name	Description
Function name	i2c_transfer_addr_set
Function prototype	void i2c_transfer_addr_set(i2c_type *i2c_x, uint16_t address);
Function description	Set master transfer address (slave address)
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.12 i2c_transfer_addr_get function

The table below describes the function i2c_transfer_addr_get.

Table 247. i2c_transfer_addr_get function

Name	Description
Function name	i2c_transfer_addr_get
Function prototype	uint16_t i2c_transfer_addr_get(i2c_type *i2c_x);
Function description	Get slave address sent from master
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
Output parameter	NA
Return value	uint16_t: slave address sent from master
Required preconditions	NA
Called functions	NA

Example:

```
i2c_transfer_addr_get(I2C1);
```

5.11.13 i2c_transfer_dir_set function

The table below describes the function i2c_transfer_dir_set.

Table 248. i2c_transfer_dir_set function

Name	Description
Function name	i2c_transfer_dir_set
Function prototype	void i2c_transfer_dir_set(i2c_type *i2c_x, i2c_transfer_dir_type i2c_direction);
Function description	Set master data transfer direction
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.14 i2c_transfer_dir_get function

The table below describes the function i2c_transfer_dir_get.

Table 249. i2c_transfer_dir_get function

Name	Description
Function name	i2c_transfer_dir_get
Function prototype	i2c_transfer_dir_type i2c_transfer_dir_get(i2c_type *i2c_x);
Function description	Get slave data transfer direction
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.15 i2c_matched_addr_get function

The table below describes the function i2c_matched_addr_get.

Table 250. i2c_matched_addr_get function

Name	Description
Function name	i2c_matched_addr_get
Function prototype	uint8_t i2c_matched_addr_get(i2c_type *i2c_x);
Function description	Get slave address match value
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
Output parameter	NA
Return value	uint8_t: slave matched address
Required preconditions	NA
Called functions	NA

Example:

```
i2c_matched_addr_get(I2C1);
```

5.11.16 i2c_auto_stop_enable function

The table below describes the function i2c_auto_stop_enable.

Table 251. i2c_auto_stop_enable function

Name	Description
Function name	i2c_auto_stop_enable
Function prototype	void i2c_auto_stop_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable auto transmit stop conditions
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.17 i2c_reload_enable function

The table below describes the function i2c_reload_enable.

Table 252. i2c_reload_enable function

Name	Description
Function name	i2c_reload_enable
Function prototype	void i2c_reload_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable transmitted data reload mode
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.18 i2c_cnt_set function

The table below describes the function i2c_cnt_set.

Table 253. i2c_cnt_set function

Name	Description
Function name	i2c_cnt_set
Function prototype	void i2c_cnt_set(i2c_type *i2c_x, uint8_t cnt);
Function description	Set the number of data to send or receive, ranging from 1 to 255
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.19 i2c_addr10_header_enable function

The table below describes the function i2c_addr10_header_enable.

Table 254. i2c_addr10_header_enable function

Name	Description
Function name	i2c_addr10_header_enable
Function prototype	void i2c_addr10_header_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable 10-bit address header read timing
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.20 i2c_general_call_enable function

The table below describes the function i2c_dma_enable.

Table 255. i2c_general_call_enable function

Name	Description
Function name	i2c_general_call_enable
Function prototype	void i2c_general_call_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable broadcast address. After enabled, broadcast address 0x00 is responded
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```


5.11.21 i2c_smbus_alert_set function

The table below describes the function i2c_smbus_alert_set.

Table 256. i2c_smbus_alert_set function

Name	Description
Function name	i2c_smbus_alert_set
Function prototype	void i2c_smbus_alert_set(i2c_type *i2c_x, i2c_smbus_alert_set_type level);
Function description	Set SMBus alert pin level (high or low)
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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

source

SMBus alert pin level

I2C_SMBUS_ALERT_LOW: SMBus alert pin output low

I2C_SMBUS_ALERT_HIGH: SMBus alert pin output high

Example:

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

5.11.22 i2c_slave_data_ctrl_enable function

The table below describes the function i2c_start_generate.

Table 257. i2c_start_generate function

Name	Description
Function name	i2c_slave_data_ctrl_enable
Function prototype	void i2c_slave_data_ctrl_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable slave data receive control. This function is used to control ACK or NACK response to each received byte when in slave receive mode. It is usually used for SMBus.
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.23 i2c_pec_calculate_enable function

The table below describes the function i2c_pec_calculate_enable

Table 258. i2c_pec_calculate_enable

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

5.11.24 i2c_pec_transmit_enable function

The table below describes the function i2c_pec_transmit_enable.

Table 259. i2c_pec_transmit_enable function

Name	Description
Function name	i2c_pec_transmit_enable
Function prototype	void i2c_pec_transmit_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	PEC transmit enable (send/receive PEC)
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.25 i2c_pec_value_get function

The table below describes the function i2c_pec_value_get

Table 260. i2c_pec_value_get function

Name	Description
Function name	i2c_pec_value_get
Function prototype	uint8_t i2c_pec_value_get(i2c_type *i2c_x);
Function description	Get current PEC value
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
Output parameter	uint8_t: current PEC value
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
pec_value = i2c_pec_value_get(I2C1);
```

5.11.26 i2c_timeout_set function

The table below describes the function i2c_timeout_set.

Table 261. i2c_timeout_set function

Name	Description
Function name	i2c_timeout_set
Function prototype	void i2c_timeout_set(i2c_type *i2c_x, uint16_t timeout);
Function description	Set SCL line level timeout detect time
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.27 i2c_timeout_detcet_set function

The table below describes the function i2c_timeout_detcet_set.

Table 262. i2c_timeout_detcet_set function

Name	Description
Function name	i2c_timeout_detcet_set
Function prototype	void i2c_timeout_detcet_set(i2c_type *i2c_x, i2c_timeout_detcet_type mode);
Function description	Set SCL line level timeout detect mode
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.28 i2c_timeout_enable function

The table below describes the function i2c_timeout_enable.

Table 263. i2c_timeout_enable function

Name	Description
Function name	i2c_timeout_enable
Function prototype	void i2c_timeout_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable SCL line level timeout detect
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.29 i2c_ext_timeout_set function

The table below describes the function i2c_ext_timeout_set.

Table 264. i2c_ext_timeout_set function

Name	Description
Function name	i2c_ext_timeout_set
Function prototype	void i2c_ext_timeout_set(i2c_type *i2c_x, uint16_t timeout);
Function description	Set SCL line cumulative clock stretching timeout value, usually used in SMBus mode
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.30 i2c_ext_timeout_enable function

The table below describes the function i2c_ext_timeout_enable.

Table 265. i2c_ext_timeout_enable function

Name	Description
Function name	i2c_ext_timeout_enable
Function prototype	void i2c_ext_timeout_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable SCL line cumulative clock stretching timeout
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.31 i2c_interrupt_enable function

The table below describes the function i2c_interrupt_enable.

Table 266. i2c_interrupt_enable function

Name	Description
Function name	i2c_interrupt_enable
Function prototype	void i2c_interrupt_enable(i2c_type *i2c_x, uint32_t source, confirm_state new_state);
Function description	I2C interrupt enable
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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

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);
```

5.11.32 i2c_interrupt_get function

The table below describes the function i2c_interrupt_get.

Table 267. i2c_interrupt_get function

Name	Description
Function name	i2c_interrupt_get
Function prototype	flag_status i2c_interrupt_get(i2c_type *i2c_x, uint16_t source);
Function description	Get interrupt enable state
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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

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);
```

5.11.33 i2c_dma_enable function

The table below describes the function i2c_dma_enable.

Table 268. i2c_dma_enable function

Name	Description
Function name	i2c_dma_enable
Function prototype	void i2c_dma_enable(i2c_type *i2c_x, i2c_dma_request_type dma_req, confirm_state new_state);
Function description	DMA transfer enable
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.34 i2c_transmit_set function

The table below describes the function i2c_transmit_set.

Table 269. i2c_transmit_set function

Name	Description
Function name	i2c_transmit_set
Function prototype	void i2c_transmit_set(i2c_type *i2c_x, uint16_t address, uint8_t cnt, i2c_reload_stop_mode_type rld_stop, i2c_start_mode_type start);
Function description	Set master transmit. This function is used to start data transfer on bus.
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
Input parameter 2	Address: slave address
Input parameter 3	Cnt: count of data to send/receive
Input parameter 4	rld_stop: reload mode and STOP condition generation mode Refer to the following “rld_stop” descriptions for details.
Input parameter 5	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);
```

5.11.35 i2c_start_generate function

The table below describes the function i2c_start_generate.

Table 270. i2c_slave_transmit function

Name	Description
Function name	i2c_start_generate
Function prototype	void i2c_start_generate(i2c_type *i2c_x);
Function description	Generate a START condition (for master)
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_start_generate(I2C1);
```

5.11.36 i2c_stop_generate function

The table below describes the function i2c_stop_generate.

Table 271. i2c_stop_generate function

Name	Description
Function name	i2c_stop_generate
Function prototype	void i2c_stop_generate(i2c_type *i2c_x);
Function description	Generate a STOP condition
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_stop_generate(I2C1);
```

5.11.37 i2c_data_send function

The table below describes the function i2c_data_send.

Table 272. i2c_data_send function

Name	Description
Function name	i2c_data_send
Function prototype	void i2c_data_send(i2c_type *i2c_x, uint8_t data);
Function description	Send data
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.38 i2c_data_receive function

The table below describes the function i2c_data_receive

Table 273. i2c_data_receive function

Name	Description
Function name	i2c_data_receive
Function prototype	uint8_t i2c_data_receive(i2c_type *i2c_x);
Function description	Receive data
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
Output parameter	uint8_t: data to be received
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
data_value = i2c_data_receive(I2C1);
```

5.11.39 i2c_flag_get function

The table below describes the function i2c_flag_get

Table 274. i2c_flag_get function

Name	Description
Function name	i2c_flag_get
Function prototype	flag_status i2c_flag_get(i2c_type *i2c_x, uint32_t flag);
Function description	Get flag status
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.40 i2c_interrupt_flag_get function

The table below describes the function i2c_interrupt_flag_get.

Table 275. i2c_interrupt_flag_get function

Name	Description
Function name	i2c_interrupt_flag_get
Function prototype	flag_status i2c_interrupt_flag_get(i2c_type *i2c_x, uint32_t flag);
Function description	Get flag status and judge the corresponding interrupt enable bit
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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 falg

Example

```
i2c_interrupt_flag_get(I2C1, I2C_TDIS_FLAG);
```

5.11.41 i2c_flag_clear function

The table below describes the function i2c_flag_clear.

Table 276. i2c_flag_clear function

Name	Description
Function name	i2c_flag_clear
Function prototype	void i2c_flag_clear(i2c_type *i2c_x, uint32_t flag);
Function description	Clear flag
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2
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);
```

5.11.42 i2c_wakeup_enable function

The table below describes the function i2c_wakeup_enable.

Table 277. i2c_wakeup_enable function

Name	Description
Function name	i2c_wakeup_enable
Function prototype	void i2c_wakeup_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable Deepsleep mode wakeup
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1
Input parameter 2	new_state: Deepsleep mode wakeup enable state This parameter can be TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
i2c_wakeup_enable(I2C1, TRUE);
```

5.11.43 i2c_analog_filter_enable function

The table below describes the function i2c_analog_filter_enable.

Table 278. i2c_analog_filter_enable function

Name	Description
Function name	i2c_analog_filter_enable
Function prototype	void i2c_analog_filter_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable analog filter
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1
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_analog_filter_enable(I2C1, TRUE);
```

5.11.44 i2c_config function

The table below describes the function i2c_config.

Table 279. 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 Refer to " i2c_handle_type " below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

i2c_handle_type* hi2c

i2c_handle_type is defined in the i2c_application.h.

typedef struct

```
{
    i2c_type          *i2cx;
    uint8_t           *pbuff;
    __IO uint16_t      psize;
    __IO uint16_t      pcount;
    __IO uint32_t      mode;
    __IO uint32_t      timeout;
    __IO uint32_t      status;
    __IO i2c_status_type error_code;
    dma_channel_type   *dma_tx_channel;
    dma_channel_type   *dma_rx_channel;
    dma_init_type      dma_init_struct;
}i2c_handle_type;
```

i2cx

Select an 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.

<code>I2C_OK:</code>	Communication OK
<code>I2C_ERR_STEP_1:</code>	Step 1 error
<code>I2C_ERR_STEP_2:</code>	Step 2 error
<code>I2C_ERR_STEP_3:</code>	Step 3 error
<code>I2C_ERR_STEP_4:</code>	Step 4 error
<code>I2C_ERR_STEP_5:</code>	Step 5 error
<code>I2C_ERR_STEP_6:</code>	Step 6 error
<code>I2C_ERR_STEP_7:</code>	Step 7 error
<code>I2C_ERR_STEP_8:</code>	Step 8 error
<code>I2C_ERR_STEP_9:</code>	Step 9 error
<code>I2C_ERR_STEP_10:</code>	Step 10 error
<code>I2C_ERR_STEP_11:</code>	Step 11 error
<code>I2C_ERR_STEP_12:</code>	Step 12 error
<code>I2C_ERR_TCRLD:</code>	Wait for TCRLD timeout
<code>I2C_ERR_TDC:</code>	Wait for TDC timeout
<code>I2C_ERR_ADDR:</code>	Address send error
<code>I2C_ERR_STOP:</code>	STOP condition send error
<code>I2C_ERR_ACKFAIL:</code>	Acknowledge error
<code>I2C_ERR_TIMEOUT:</code>	Timeout error
<code>I2C_ERR_INTERRUPT:</code>	Enter an interrupt when an error event occurred

dma_tx_channel

I2C transmit DMA channel

dma_rx_channel

I2C receive DMA channel

dma_init_struct

DMA initialization structure

Example:

```
i2c_handle_type hi2c;
hi2c.i2cx = I2C1;
i2c_config(&hi2c);
```


5.11.45 i2c_lowlevel_init function

The table below describes the function i2c_lowlevel_init.

Table 280. 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” in section 5.11.44 i2c_config function for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
void i2c_lowlevel_init(i2c_handle_type* hi2c)
{
    if(hi2c->i2cx == I2C1)
    {
        Implement I2C1 initialization
    }
    else if(hi2c->i2cx == I2C2)
    {
        Implement I2C1 initialization
    }
}
```

5.11.46 i2c_wait_end function

The table below describes the function i2c_wait_end.

Table 281. 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” in section 5.11.44 i2c_config function for details.
Input parameter 2	Timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```

if (i2c_master_transmit_dma(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF) != I2C_OK)
{
    error_handler(i2c_status);
}

/* wait for the end of transfer*/
if(i2c_wait_end(&hi2c, 0xFFFFFFFF) != I2C_OK)
{
    error_handler(i2c_status);
}

```

5.11.47 i2c_wait_flag function

The table below describes the function i2c_wait_flag.

Table 282. 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” in section 5.11.44 i2c_config function for details.
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 “event_check” descriptions below for details.
Input parameter 4	Timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.11.44 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);
```

5.11.48 i2c_master_transmit function

The table below describes the function i2c_master_transmit.

Table 283. 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” in section 5.11.44 i2c_config function for details.
Input parameter 2	Address: slave address
Input parameter 3	Pdata: array address of to-be-sent data
Input parameter 4	Size: the size of data to be sent
Input parameter 5	Timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_transmit(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

5.11.49 i2c_master_receive function

The table below describes the function i2c_master_receive.

Table 284. i2c_master_receivefunction

Name	Description
Function name	i2c_master_receive
Function prototype	i2c_status_type i2c_master_receive(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Master receives data (polling mode). This function is a blocking type. After the execution is done, so does I2C transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to "i2c_handle_type" in section 5.11.44 i2c_config function for details.
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 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_receive(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

5.11.50 i2c_slave_transmit function

The table below describes the function i2c_slave_transmit.

Table 285. i2c_slave_transmit function

Name	Description
Function name	i2c_slave_transmit
Function prototype	i2c_status_type i2c_slave_transmit(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave sends data (polling mode). This function is a blocking type. In other words, after the function execution is done, so is I2C transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to "i2c_handle_type" in section 5.11.44 i2c_config function for details.
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 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_transmit(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

5.11.51 i2c_slave_receive function

The table below describes the function i2c_slave_receive.

Table 286. i2c_slave_receive function

Name	Description
Function name	i2c_slave_receive
Function prototype	i2c_status_type i2c_slave_receive(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave receives data (polling mode). This function is a blocking type. In other words, after the function execution is done, so is I2C transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to "i2c_handle_type" in section 5.11.44 i2c_config function for details.
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 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_receive(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

5.11.52 i2c_master_transmit_int function

The table below describes the function i2c_master_transmit_int.

Table 287. 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" in section 5.11.44 i2c_config function for details.
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 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_transmit_int(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

5.11.53 i2c_master_receive_int function

The table below describes the function i2c_master_receive_int.

Table 288. 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 the i2c_wait_end() to wait for the end of transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to "i2c_handle_type" in section 5.11.44 i2c_config function for details.
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 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_receive_int(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```


5.11.54 i2c_slave_transmit_int function

The table below describes the function i2c_master_receive_int.

Table 289. i2c_master_receive_int function

Name	Description
Function name	i2c_slave_transmit_int
Function prototype	i2c_status_type i2c_slave_transmit_int(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave sends data (through interrupt mode). This function operates in non-blocking mode. In other words, after the function is executed, the 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” in section 5.11.44 i2c_config function for details.
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 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_transmit_int(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

5.11.55 i2c_slave_receive_int function

The table below describes the function i2c_slave_receive_int

Table 290. i2c_master_receive_int function

Name	Description
Function name	i2c_slave_receive_int
Function prototype	i2c_status_type i2c_slave_receive_int(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave receives data (through interrupt mode). This function is a non-blocking type. In other words, after the function is executed, the 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” in section 5.11.44 i2c_config function for details.
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 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_receive_int(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

5.11.56 i2c_master_transmit_dma function

The table below describes the function `i2c_master_transmit_dma`.

Table 291. i2c_master_transmit_dma function

Name	Description
Function name	<code>i2c_master_transmit_dma</code>
Function prototype	<code>i2c_status_type i2c_master_transmit_dma(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);</code>
Function description	Master 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 <code>i2c_wait_end()</code> to wait for the end of transfer.
Input parameter 1	hi2c: <code>i2c_handle_type</code> pointer Refer to “ <code>i2c_handle_type</code> ” in section 5.11.44 i2c_config function for details.
Input parameter 2	Address: slave address
Input parameter 3	Pdata: array address of data to be sent
Input parameter 4	Size: number of data to send
Input parameter 5	Timeout: wait timeout
Output parameter	NA
Return value	<code>i2c_status_type</code> : error code Refer to 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_transmit_dma(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

5.11.57 i2c_master_receive_dma function

The table below describes the function i2c_master_receive_dma.

Table 292. 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" in section 5.11.44 i2c_config function for details.
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 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_master_receive_dma(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

5.11.58 i2c_slave_transmit_dma function

The table below describes the function i2c_slave_transmit_dma.

Table 293. 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" in section 5.11.44 i2c_config function for details.
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 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_transmit_dma(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

5.11.59 i2c_slave_receive_dma function

The table below describes the function i2c_slave_transmit_dma.

Table 294. 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.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to "i2c_handle_type" in section 5.11.44 i2c_config function for details.
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 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_slave_receive_dma(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

5.11.60 i2c_smbus_master_transmit function

The table below describes the function i2c_smbus_master_transmit

Table 295. i2c_smbus_master_transmit function

Name	Description
Function name	i2c_smbus_master_transmit
Function prototype	i2c_status_type i2c_smbus_master_transmit(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	SMBus master sends data (through polling mode). This function is a blocking type. In other words, after the function execution is done, so is data transfer. It is mainly used for PEC transmission and reception.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to "i2c_handle_type" in section 5.11.44 i2c_config function for details.
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 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_smbus_master_transmit(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

5.11.61 i2c_smbus_master_receive function

The table below describes the function i2c_smbus_master_receive

Table 296. i2c_smbus_master_receive function

Name	Description
Function name	i2c_smbus_master_receive
Function prototype	i2c_status_type i2c_smbus_master_receive(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	SMBus master receives data (through polling mode). This function is a blocking type. In other words, after the function execution is done, so is data transfer. It is mainly used for PEC transmission and reception.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to "i2c_handle_type" in section 5.11.44 i2c_config function for details.
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 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_smbus_master_receive(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

5.11.62 i2c_smbus_slave_transmit function

The table below describes the function i2c_smbus_slave_transmit

Table 297. i2c_smbus_slave_transmit function

Name	Description
Function name	i2c_smbus_slave_transmit
Function prototype	i2c_status_type i2c_smbus_slave_transmit(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	SMBus slave sends data (through polling mode). This function is a blocking type. In other words, after the function execution is done, so is data transfer. It is mainly used for PEC transmission and reception.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to "i2c_handle_type" in section 5.11.44 i2c_config function for details.
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 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_smbus_slave_transmit(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```


5.11.63 i2c_smbus_slave_receive function

The table below describes the function i2c_smbus_slave_receive

Table 298. 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" in section 5.11.44 i2c_config function for details.
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 5.11.44 for details.
Required preconditions	NA
Called functions	NA

Example:

```
i2c_smbus_slave_receive(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

5.11.64 i2c_memory_write function

The table below describes the function i2c_memory_write

Table 299. i2c_memory_write function

Name	Description
Function name	i2c_memory_write
Function prototype	i2c_status_type i2c_memory_write(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Write data to EEPROM (through polling mode). This function is a blocking type. In other words, after the function execution is done, so is I2C transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to "i2c_handle_type" in section 5.11.44 i2c_config function for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the "mem_address_width" below for details.
Input parameter 3	address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	Pdata: array address of data to be sent
Input parameter 6	Size: number of data to be sent
Input parameter 7	Timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.11.44 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);
```

5.11.65 i2c_memory_write_int function

The table below describes the function i2c_memory_write_int

Table 300. 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" in section 5.11.44 i2c_config function for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the "mem_address_width" below for details.
Input parameter 3	address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	pdata: array address of data to be sent
Input parameter 6	size: number of data to be sent
Input parameter 7	Timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.11.44 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);
```

5.11.66 i2c_memory_write_dma function

The table below describes the function i2c_memory_write_dma

Table 301. 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" in section 5.11.44 i2c_config function for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the "mem_address_width" below for details.
Input parameter 3	address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	pdata: array address of data to be sent
Input parameter 6	size: number of data to be sent
Input parameter 7	Timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.11.44 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);
```

5.11.67 i2c_memory_read function

The table below describes the function i2c_memory_write_dma

Table 302. i2c_memory_write_dma 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 DMA mode). This function is a blocking type. In other words, after the function execution is done, so is data transfer. It is mainly used for PEC transmission and reception.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to “i2c_handle_type” in section 5.11.44 i2c_config function for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the “mem_address_width” below for details.
Input parameter 3	address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	pdata: array address of data to be read
Input parameter 6	size: number of data to be read
Input parameter 7	Timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to 5.11.44 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);
```

5.11.68 i2c_memory_read_int function

The table below describes the function `i2c_memory_read_int`

Table 303. i2c_memory_write_dma function

Name	Description
Function name	<code>i2c_memory_read_int</code>
Function prototype	<code>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);</code>
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 <code>i2c_wait_end()</code> to wait for the end of transfer
Input parameter 1	hi2c: <code>i2c_handle_type</code> pointer Refer to “ <code>i2c_handle_type</code> ” in section 5.11.44 i2c_config function for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the “mem_address_width” below for details.
Input parameter 3	address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	pdata: array address of data to be read
Input parameter 6	size: number of data to be read
Input parameter 7	Timeout: wait timeout
Output parameter	NA
Return value	<code>i2c_status_type</code> : error code Refer to 5.11.44 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);
```

5.11.69 i2c_memory_read_dma function

The table below describes the function `i2c_memory_read_dma`

Table 304. i2c_memory_write_dma function

Name	Description
Function name	<code>i2c_memory_read_dma</code>
Function prototype	<code>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);</code>
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. So in this case, it is possible to call the <code>i2c_wait_end()</code> to wait for the end of transfer
Input parameter 1	hi2c: <code>i2c_handle_type</code> pointer Refer to “ <code>i2c_handle_type</code> ” in section 5.11.44 i2c_config function for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the “mem_address_width” below for details.
Input parameter 3	address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	pdata: array address of data to be read
Input parameter 6	size: number of data to be read
Input parameter 7	Timeout: wait timeout
Output parameter	NA
Return value	<code>i2c_status_type</code> : error code Refer to 5.11.44 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);
```

5.11.70 i2c_evt_irq_handler function

The table below describes the function i2c_evt_irq_handler

Table 305. 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" in section 5.11.44 i2c_config function for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
void I2C1_EVT_IRQHandler(void)
{
    i2c_evt_irq_handler(&hi2c);
}
```

5.11.71 i2c_err_irq_handler function

The table below describes the function i2c_err_irq_handler

Table 306. 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" in section 5.11.44 i2c_config function for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
void I2C1_ERR_IRQHandler(void)
{
    i2c_err_irq_handler(&hi2c);
}
```


5.11.72 i2c_dma_tx_irq_handler function

The table below describes the function i2c_dma_tx_irq_handler

Table 307. 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" in section 5.11.44 i2c_config function for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
void DMA1_Channel6_IRQHandler(void)
{
    i2c_dma_tx_irq_handler(&hi2c);
}
```

5.11.73 i2c_dma_rx_irq_handler function

The table below describes the function i2c_dma_rx_irq_handler

Table 308. 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" in section 5.11.44 i2c_config function for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
void DMA1_Channel7_IRQHandler(void)
{
    i2c_dma_rx_irq_handler(&hi2c);
}
```

5.12 Nested vectored interrupt controller (NVIC)

The NVIC register structure NVIC_Type is defined in the “core_cm0plus.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 309. 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 310. Summary of NVIC library functions

Function name	Description
nvic_system_reset	System software reset
nvic_irq_enable	NVIC interrupt enable and priority enable
nvic_irq_disable	NVIC interrupt disable
nvic_priority_group_config	NVIC interrupt priority grouping configuration
nvic_vector_table_set	NVIC interrupt vector table base address and offset address configuration
nvic_lowpower_mode_config	NVIC low-power mode configuration

5.12.1 nvic_system_reset function

The table below describes the function nvic_system_reset.

Table 311. nvic_system_reset function

Name	Description
Function name	nvic_system_reset
Function prototype	void nvic_system_reset(void)
Function description	System software reset
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NVIC_SystemReset()

Example:

```
/* system reset */
nvic_system_reset();
```

5.12.2 nvic_irq_enable function

The table below describes the function nvic_irq_enable.

Table 312. 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 the “ irqn ” descriptions below for details.
Input parameter 2	preempt_priority: set preemption priority This parameter must not be greater than the highest preemption priority defined in the NVIC_PRIORITY_GROUP_x
Input parameter 3	sub_priority: set response priority This parameter is not supported in L0 series, please ignore.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NVIC_SetPriority() NVIC_EnableIRQ()

irqn

irqn is used to select interrupt vectors, including:

WWDT_IRQn:	Window timer interrupt
PVM_IRQn:	PVM interrupt linked to EXINT
.....	
USART4_3_IRQn:	USART3&4 interrupt
CAN1_IRQn:	CAN1 interrupt

Example:

```
/* enable nvic irq */
nvic_irq_enable(ADC1_IRQn, 0, 0);
```

5.12.3 nvic_irq_disable function

The table below describes the function nvic_irq_disable.

Table 313. 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 <i>irqn</i> for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NVIC_DisableIRQ()

Example:

```
/* disable nvic irq */
nvic_irq_disable(ADC1_IRQn);
```

5.12.4 nvic_vector_table_set function

The table below describes the function nvic_vector_table_set.

Table 314. 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

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);
```

5.12.5 nvic_lowpower_mode_config function

The table below describes the function nvic_lowpower_mode_config.

Table 315. nvic_lowpower_mode_config function

Name	Description
Function name	nvic_lowpower_mode_config
Function prototype	void nvic_lowpower_mode_config(nvic_lowpower_mode_type lp_mode, confirm_state new_state)
Function description	Configure NVIC low-power mode
Input parameter 1	lp_mode: select low-power modes This parameter can be any enumerated value in the nvic_lowpower_mode_type.
Input parameter 2	new_state: indicates the pre-configured status of battery powered domain This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

lp_mode

lp_mode is used to select low-power modes, including:

NVIC_LP_SEVONPEND:

Send wakeup event upon interrupt suspend (this option is usually used in conjunction with WFE)

NVIC_LP_SLEEPDEEP:

Deepsleep mode control bit (enable or disable core clock)

NVIC_LP_SLEEPONEXIT: Sleep mode entry when system leaves the lowest-priority interrupt

Example:

```
/* enable sleep-on-exit feature */  
nvic_lowpower_mode_config(NVIC_LP_SLEEPONEXIT, TRUE);
```

5.13 System configuration controller (SCFG)

The SCFG register structure `scfg_type` is defined in the “at32l021_scfg.h”

```
/**
 * @brief type define scfg register all
 */
typedef struct
{
    ...
} scfg_type;
```

The table below gives a list of the SCFG registers

Table 316. Summary of SCFG registers

Register	Description
<code>scfg_cfg1</code>	SCFG configuration register 1
<code>scfg_exintc1</code>	SCFG external interrupt configuration register 1
<code>scfg_exintc2</code>	SCFG external interrupt configuration register 2
<code>scfg_exintc3</code>	SCFG external interrupt configuration register 3
<code>scfg_exintc4</code>	SCFG external interrupt configuration register 4
<code>scfg_cfg2</code>	SCFG configuration register 2

The table below gives a list of SCFG library functions.

Table 317. Summary of SCFG library functions

Function name	Description
<code>scfg_reset</code>	SCFG reset
<code>scfg_infrared_config</code>	infrared configuration
<code>scfg_mem_map_get</code>	Get memory address map
<code>scfg_pa11pa12_pin_remap</code>	PA11 and PA12 remap
<code>scfg_exint_line_config</code>	External interrupt line configuration
<code>scfg_sram_operr_lock_enable</code>	Enable SRAM parity check error lock
<code>scfg_pins_ultra_driven_enable</code>	Pin ultra-high current sinking capability enable
<code>scfg_pvm_lock_enable</code>	PVM Lock enable
<code>scfg_sram_operr_status_get</code>	Get SRAM parity check error status
<code>scfg_lockup_enable</code>	Lockup lock enable

5.13.1 scfg_reset function

The table below describes the function scfg_reset.

Table 318. scfg_reset function

Name	Description
Function name	scfg_reset
Function prototype	void scfg_reset(void);
Function description	Reset SCFG
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
scfg_reset();
```

5.13.2 scfg_infrared_config function

The table below describes the function scfg_infrared_config.

Table 319. scfg_infrared_config function

Name	Description
Function name	scfg_infrared_config
Function prototype	void scfg_infrared_config(scfg_ir_source_type source, scfg_ir_polarity_type polarity);
Function description	Infrared configuration
Input parameter 1	Source: infrared modulation signal source
Input parameter 2	Polarity: output signal polarity
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

scfg_ir_source_type

Select infrared signal source.

SCFG_IR_SOURCE_TMR10: Infrared signal source is tmr10

scfg_ir_polarity_type

Select infrared signal polarity.

SCFG_IR_POLARITY_NO_AFFECTE: Infrared output signal not inverted

SCFG_IR_POLARITY_REVERSE: Infrared output signal inverted

Example:

```
scfg_infrared_config(SCFG_IR_SOURCE_TMR16, SCFG_IR_POLARITY_NO_AFFECTE);
```

5.13.3 scfg_mem_map_get function

The table below describes the function `scfg_mem_map_get`.

Table 320. scfg_mem_map_get function

Name	Description
Function name	<code>scfg_mem_map_get</code>
Function prototype	<code>scfg_mem_map_type scfg_mem_map_get(void);</code>
Function description	Get memory address mapping
Input parameter	NA
Output parameter	NA
Return value	<code>scfg_mem_map_type</code> : return memory address mapping option
Required preconditions	NA
Called functions	NA

scfg_mem_map_type

Memory address mapping type.

SCFG_MEM_MAP_MAIN_MEMORY: Main memory is mapped to 0x00000000

SCFG_MEM_MAP_BOOT_MEMORY: Boot memory is mapped to 0x00000000

SCFG_MEM_MAP_INTERNAL_SRAM: Internal memory is mapped to 0x00000000

Example:

```
scfg_mem_map_type mem_map = scfg_mem_map_get();
```

5.13.4 scfg_pa11pa12_pin_remap function

The table below describes the function `scfg_pa11pa12_pin_remap`.

Table 321. scfg_pa11pa12_pin_remap

Name	Description
Function name	<code>scfg_pa11pa12_pin_remap</code>
Function prototype	<code>void scfg_pa11pa12_pin_remap(scfg_pa11pa12_remap_type pin_remap);</code>
Function description	Remap PA11 and PA12
Input parameter	<code>pin_remap</code> : remap option
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

scfg_pa11pa12_remap_type

Pin remap options.

SCFG_PA11PA12_NO_REMAP: PA11 and PA12 are not remapped

SCFG_PA11PA12_TO_PA9PA10: PA11 and PA12 are remapped to PA9 and PA10

Example:

```
scfg_pa11pa12_pin_remap(SCFG_PA11PA12_TO_PA9PA10);
```


5.13.5 scfg_exint_line_config function

The table below describes the function `scfg_exint_line_config`.

Table 322. scfg_exint_line_config

Name	Description
Function name	<code>scfg_exint_line_config</code>
Function prototype	<code>void scfg_exint_line_config(scfg_port_source_type port_source, scfg_pins_source_type pin_source);</code>
Function description	Configure external interrupt line
Input parameter 1	<code>port_source</code> : port source
Input parameter 2	<code>pin_source</code> : pin source
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

scfg_port_source_type

SCFG_PORT_SOURCE_GPIOA: port A

SCFG_PORT_SOURCE_GPIOB: port B

SCFG_PORT_SOURCE_GPIOC: port C

SCFG_PORT_SOURCE_GPIOF: port F

scfg_pins_source_type

SCFG_PINS_SOURCE0: pin 0

SCFG_PINS_SOURCE1: pin 1

SCFG_PINS_SOURCE2: pin 2

.....

SCFG_PINS_SOURCE13: pin 13

SCFG_PINS_SOURCE14: pin 14

SCFG_PINS_SOURCE15: pin 15

Example:

```
scfg_exint_line_config(SCFG_PORT_SOURCE_GPIOA, SCFG_PINS_SOURCE1);
```

5.13.6 scfg_sram_operr_lock_enable function

The table below describes the function `scfg_sram_operr_lock_enable`.

Table 323. scfg_sram_operr_lock_enable function

Name	Description
Function name	<code>scfg_sram_operr_lock_enable</code>
Function prototype	<code>void scfg_sram_operr_lock_enable(confirm_state new_state);</code>
Function description	Enable SRAM parity check error lock
Input parameter	<code>new_state</code> : enabled or disabled This parameter can be FALSE or TRUE
Output parameter	NA
Return value	<code>error_status</code> : return SRAM parity check error status Normal (SUCCESS), Error (ERROR)
Required preconditions	NA
Called functions	NA

Example:

```
scfg_sram_operr_lock_enable(TRUE);
```

5.13.7 scfg_pins_ultra_driven_enable function

The table below describes the function `scfg_pins_ultra_driven_enable`.

Table 324. scfg_pins_ultra_driven_enable function

Name	Description
Function name	<code>scfg_pins_ultra_driven_enable</code>
Function prototype	<code>void scfg_pins_ultra_driven_enable(scfg_ultra_driven_pins_type value, confirm_state new_state);</code>
Function description	Enable pin ultra-high current sinking capability
Input parameter 1	Value: pin
Input parameter 2	<code>new_state</code> : enabled or disabled 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

Example:

```
scfg_pins_ultra_driven_enable(SCFG_ULTRA_DRIVEN_PB8, TRUE);
```

5.13.8 scfg_pvm_lock_enable function

The table below describes the function `scfg_adc_dma_channel_remap`.

Table 325. scfg_adc_dma_channel_remap function

Name	Description
Function name	<code>scfg_pvm_lock_enable</code>
Function prototype	<code>void scfg_pvm_lock_enable(confirm_state new_state);</code>
Function description	PVM Lock enable
Input parameter	<code>new_state</code> : enabled or disables This parameter can be FALSE or TRUE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
scfg_pvm_lock_enable(TRUE);
```

5.13.9 scfg_sram_operr_status_get function

The table below describes the function `scfg_sram_operr_status_get`.

Table 326. scfg_sram_operr_status_get function

Name	Description
Function name	<code>scfg_sram_operr_status_get</code>
Function prototype	<code>error_status scfg_sram_operr_status_get(void);</code>
Function description	Get parity check error status
Input parameter	<code>new_state</code> : enabled or disables This parameter can be FALSE or TRUE
Output parameter	NA
Return value	<code>error_status</code> : return SRAM parity check error status Normal (SUCCESS), Error (ERROR)
Required preconditions	NA
Called functions	NA

Example:

```
error_status status = scfg_sram_operr_status_get();
```

5.13.10 scfg_lockup_enable function

The table below describes the function `scfg_uart1_tx_dma_channel_remap`.

Table 327. scfg_uart1_tx_dma_channel_remap function

Name	Description
Function name	<code>scfg_lockup_lock_enable</code>
Function prototype	<code>void scfg_lockup_lock_enable(confirm_state new_state);</code>
Function description	Lockup lock enable
Input parameter	<code>new_state</code> : enabled or disables This parameter can be FALSE or TRUE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
scfg_lockup_lock_enable(TRUE);
```

5.14 Serial peripheral interface (SPI)/ I²S

The SPI register structure spi_type is defined in the “at32l021_spi.h”:

```
/**
 * @brief type define spi register all
 */
typedef struct
{
    ...
} spi_type;
```

The table below gives a list of the SPI registers

Table 328. Summary of SPI registers

Register	Description
ctrl1	SPI control register 1
ctrl2	SPI control register 2
sts	SPI status register
dt	SPI data register
cpoly	SPI CRC register
rcrc	SPI RxCRC register
tcrc	SPI TxCRC register
i2sctrl	SPI_I2S configuration register
i2sclkp	SPI_I2S prescaler register

The table below gives a list of SPI library functions.

Table 329. Summary of SPI library functions

Function name	Description
spi_i2s_reset	Reset SPI/I ² S registers to their reset values
spi_default_para_init	Configure the SPI initialization structure with an initial value
spi_init	Initialize SPI
spi_ti_mode_enable	Enable SPI TI mode
spi_crc_next_transmit	Next data transfer is CRC command
spi_crc_polynomial_set	SPI CRC polynomial configuration
spi_crc_polynomial_get	Get SPI CRC polynomial
spi_crc_enable	Enable SPI CRC
spi_crc_value_get	Get CRC result of SPI receive/transmit
spi_hardware_cs_output_enable	Enable hardware CS output
spi_software_cs_internal_level_set	Set software CS internal level
spi_frame_bit_num_set	Set the number of frame bits
spi_half_duplex_direction_set	Set transfer direction of single-wire bidirectional half-duplex mode
spi_enable	Enable SPI
i2s_default_para_init	Set an initial value for the I ² S initialization structure
i2s_init	Initialize I ² S
i2s_enable	Enable I ² S
spi_i2s_interrupt_enable	Enable SPI/I ² S interrupts

spi_i2s_dma_transmitter_enable	Enable SPI/I ² S DMA transmit
spi_i2s_dma_receiver_enable	Enable SPI/I ² S DMA receive
spi_i2s_data_transmit	SPI/I ² S transmits data
spi_i2s_data_receive	SPI/I ² S receives data
spi_i2s_flag_get	Get SPI/I ² S flags
spi_i2s_flag_clear	Clear SPI/I ² S flags

5.14.1 spi_i2s_reset function

The table below describes the function spi_i2s_reset.

Table 330. spi_i2s_reset function

Name	Description
Function name	spi_i2s_reset
Function prototype	void spi_i2s_reset(spi_type *spi_x);
Function description	Reset SPI/I ² S registers to their reset values.
Input parameter 1	spi_x: select SPI peripherals This parameter can be SPI1, SPI2
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

Example:

```
spi_i2s_reset (SPI1);
```

5.14.2 spi_default_para_init function

The table below describes the function spi_default_para_init.

Table 331. 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 spi_init_type before starting.
Called functions	NA

Example:

```
spi_init_type spi_init_struct;
spi_default_para_init (&spi_init_struct);
```

5.14.3 spi_init function

The table below describes the function spi_init.

Table 332. spi_init function

Name	Description
Function name	spi_init
Function prototype	void spi_init(spi_type* spi_x, spi_init_type* spi_init_struct);
Function description	Initialize SPI
Input parameter 1	spi_x: select SPI peripherals This parameter can be SPI1, SPI2
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 spi_init_type before starting.
Called functions	NA

spi_init_type is defined in the at32l021_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);
```


5.14.4 spi_ti_mode_enable function

The table below describes the function spi_crc_next_transmit.

Table 333. spi_ti_mode_enable function

Name	Description
Function name	spi_ti_mode_enable
Function prototype	void spi_ti_mode_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable SPI TI mode
Input parameter 1	spi_x: select SPI peripherals This parameter can be SPI1, SPI2, SPI3, SPI4.
Input parameter 2	new_state: enabled or disabled 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);
```

5.14.5 spi_crc_next_transmit function

The table below describes the function spi_crc_next_transmit.

Table 334. spi_crc_next_transmit function

Name	Description
Function name	spi_crc_next_transmit
Function prototype	void spi_crc_next_transmit(spi_type* spi_x);
Function description	The next data to be sent is CRC command
Input parameter 1	spi_x: select SPI peripherals This parameter can be SPI1, SPI2
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
spi_crc_next_transmit (SPI1);
```

5.14.6 spi_crc_polynomial_set function

The table below describes the function spi_crc_polynomial_set.

Table 335. spi_crc_polynomial_set function

Name	Description
Function name	spi_crc_polynomial_set
Function prototype	void spi_crc_polynomial_set(spi_type* spi_x, uint16_t crc_poly);
Function description	Set SPI CRC polynomial
Input parameter 1	spi_x: select SPI peripherals This parameter can be SPI1, SPI2
Input parameter 2	crc_poly: CRC polynomial Value is 0x0000~0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/*set spi crc polynomial value */
spi_crc_polynomial_set (SPI1, 0x07);
```

5.14.7 spi_crc_polynomial_get function

The table below describes the function spi_crc_polynomial_get.

Table 336. spi_crc_polynomial_get function

Name	Description
Function name	spi_crc_polynomial_get
Function prototype	uint16_t spi_crc_polynomial_get(spi_type* spi_x);
Function description	Get SPI CRC polynomial
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Output parameter	NA
Return value	CRC polynomial Value is 0x0000~0xFFFF
Required preconditions	NA
Called functions	NA

Example:

```
/*get spi crc polynomial value */
uint16_t crc_poly;
crc_poly = spi_crc_polynomial_get (SPI1);
```

5.14.8 spi_crc_enable function

The table below describes the function spi_crc_enable.

Table 337. spi_crc_enable function

Name	Description
Function name	spi_crc_enable
Function prototype	void spi_crc_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable SPI CRC
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Input parameter 2	new_state: enabled or disabled 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);
```

5.14.9 spi_crc_value_get function

The table below describes the function spi_crc_value_get.

Table 338. spi_crc_value_get function

Name	Description
Function name	spi_crc_value_get
Function prototype	uint16_t spi_crc_value_get(spi_type* spi_x, spi_crc_direction_type crc_direction);
Function description	Get SPI receive/transmit CRC result
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
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);
```

5.14.10 spi hardware_cs_output_enable function

The table below describes the function spi hardware_cs_output_enable.

Table 339. spi hardware_cs_output_enable function

Name	Description
Function name	spi hardware_cs_output_enable
Function prototype	void spi hardware_cs_output_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable hardware CS output
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Input parameter 2	new_state: enabled or disabled This parameter can FALSE or TRUE
Output parameter	NA
Return value	NA
Required preconditions	This setting is applicable to SPI master mode only.
Called functions	NA

Example:

```
/* enable the hardware cs output */
spi hardware_cs_output_enable (SPI1, TRUE);
```

5.14.11 spi software_cs_internal_level_set function

The table below describes the function spi software_cs_internal_level_set.

Table 340. spi software_cs_internal_level_set function

Name	Description
Function name	spi software_cs_internal_level_set
Function prototype	void spi software_cs_internal_level_set(spi_type* spi_x, spi software_cs_level_type level);
Function description	Set software CS internal level
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Input parameter 2	<i>level</i> : set software CS internal level
Output parameter	NA
Return value	NA
Required preconditions	1. This setting is applicable to software CS mode only; 2. In master mode, the "level" value must be "SPI_SWCS_INTERNAL_LEVEL_HIGHT".
Called functions	NA

level

Set software CS internal level

SPI_SWCS_INTERNAL_LEVEL_LOW: Software CS internal low level

SPI_SWCS_INTERNAL_LEVEL_HIGHT: Software CS internal high level

Example:

```
/* set the internal level high */
spi software_cs_internal_level_set (SPI1, SPI_SWCS_INTERNAL_LEVEL_HIGHT);
```

5.14.12 spi_frame_bit_num_set function

The table below describes the function spi_frame_bit_num_set.

Table 341. spi_frame_bit_num_set function

Name	Description
Function name	spi_frame_bit_num_set
Function prototype	void spi_frame_bit_num_set(spi_type* spi_x, spi_frame_bit_num_type bit_num);
Function description	Set the number of bits in a frame
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Input parameter 2	<i>bit_num</i> : Set the number of bits in a frame
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

bit_num

Set the number of bits in a frame

SPI_FRAME_8BIT: 8-bit data in a frame

SPI_FRAME_16BIT: 16-bit data in a frame

Example:

```
/* set the data frame bit num as 8 */
spi_frame_bit_num_set (SPI1, SPI_FRAME_8BIT);
```

5.14.13 spi_half_duplex_direction_set function

The table below describes the function spi_half_duplex_direction_set.

Table 342. spi_half_duplex_direction_set function

Name	Description
Function name	spi_half_duplex_direction_set
Function prototype	void spi_half_duplex_direction_set(spi_type* spi_x, spi_half_duplex_direction_type direction);
Function description	Set the transfer direction of single-wire bidirectional half-duplex mode
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Input parameter 2	<i>direction</i> : transfer direction
Output parameter	NA
Return value	NA
Required preconditions	This setting is applicable to the single-wire bidirectional half-duplex mode only.
Called functions	NA

direction

Transfer direction

SPI_HALF_DUPLEX_DIRECTION_RX: Receive

SPI_HALF_DUPLEX_DIRECTION_TX: Transmit

Example:

```
/* set the data transmission direction as transmit */
spi_half_duplex_direction_set (SPI1, SPI_HALF_DUPLEX_DIRECTION_TX);
```

5.14.14 spi_enable function

The table below describes the function spi_enable.

Table 343. spi_enable function

Name	Description
Function name	spi_enable
Function prototype	void spi_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable SPI
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Input parameter 2	new_state: enabled or disabled 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);
```

5.14.15 i2s_default_para_init function

The table below describes the function i2s_default_para_init.

Table 344. i2s_default_para_init function

Name	Description
Function name	i2s_default_para_init
Function prototype	void i2s_default_para_init(i2s_init_type* i2s_init_struct);
Function description	Set an initial value for the I ² S 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 i2s_init_type before starting.
Called functions	NA

Example:

```
i2s_init_type i2s_init_struct;
i2s_default_para_init (&i2s_init_struct);
```

5.14.16 i2s_init function

The table below describes the function i2s_init.

Table 345. i2s_init function

Name	Description
Function name	i2s_init
Function prototype	void i2s_init(spi_type* spi_x, i2s_init_type* i2s_init_struct);
Function description	Initialize I ² S
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
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 i2s_init_type before starting.
Called functions	NA

i2s_init_type is defined in the at32l021_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 mode

I2S_MODE_SLAVE_TX:	I2S slave transmit
I2S_MODE_SLAVE_RX:	I2S slave receive
I2S_MODE_MASTER_TX:	I2S master transmit
I2S_MODE_MASTER_RX:	I2S master receive

audio_protocol

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 TURE.

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);
```

5.14.17 i2s_enable function

The table below describes the function i2s_enable.

Table 346. i2s_enable function

Name	Description
Function name	i2s_enable
Function prototype	void i2s_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable I ² S
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable i2s*/
i2s_enable (SPI1, TRUE);
```


5.14.18 spi_i2s_interrupt_enable function

The table below describes the function spi_i2s_interrupt_enable.

Table 347. spi_i2s_interrupt_enable function

Name	Description
Function name	spi_i2s_interrupt_enable
Function prototype	void spi_i2s_interrupt_enable(spi_type* spi_x, uint32_t spi_i2s_int, confirm_state new_state);
Function description	Enable SPI/I ² S interrupts
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Input parameter 2	spi_i2s_int : select SPI interrupts
Input parameter 3	new_state: Enable or disable This parameter can be FALSE or TURE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

spi_i2s_int

Select 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);
```

5.14.19 spi_i2s_dma_transmitter_enable function

The table below describes the function spi_i2s_dma_transmitter_enable.

Table 348. spi_i2s_dma_transmitter_enable function

Name	Description
Function name	spi_i2s_dma_transmitter_enable
Function prototype	void spi_i2s_dma_transmitter_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable SPI/I ² S DMA transmitter
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Input parameter 2	new_state: enabled or disabled 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);
```

5.14.20 spi_i2s_dma_receiver_enable function

The table below describes the function spi_i2s_dma_receiver_enable.

Table 349. spi_i2s_dma_receiver_enable function

Name	Description
Function name	spi_i2s_dma_receiver_enable
Function prototype	void spi_i2s_dma_receiver_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable SPI/I ² S DMA receiver
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Input parameter 2	new_state: enabled or disabled 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);
```

5.14.21 spi_i2s_data_transmit function

The table below describes the function spi_i2s_data_transmit.

Table 350. spi_i2s_data_transmit function

Name	Description
Function name	spi_i2s_data_transmit
Function prototype	void spi_i2s_data_transmit(spi_type* spi_x, uint16_t tx_data);
Function description	SPI/I ² S sends data
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Input parameter 2	tx_data: data to send Value range (for 8-bit bit in a frame): 0x00~0xFF Value range (for16-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);
```

5.14.22 spi_i2s_data_receive function

The table below describes the function spi_i2s_data_receive.

Table 351. spi_i2s_data_receive function

Name	Description
Function name	spi_i2s_data_receive
Function prototype	uint16_t spi_i2s_data_receive(spi_type* spi_x);
Function description	SPI/I ² S receives data
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Output parameter	rx_data: data to receive Value range (for 8-bit bit in a frame): 0x00~0xFF Value range (for16-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);
```

5.14.23 spi_i2s_flag_get function

The table below describes the function spi_i2s_flag_get.

Table 352. spi_i2s_flag_get function

Name	Description
Function name	spi_i2s_flag_get
Function prototype	flag_status spi_i2s_flag_get(spi_type* spi_x, uint32_t spi_i2s_flag);
Function description	Get SPI/I ² S flag status
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Input parameter 2	spi_i2s_flag: flag selection Refer to the “spi_i2s_flag” description below for details.
Output parameter	NA
Return value	flag_status: flag status This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

spi_i2s_flag

SPI/I²S is used to select a flag from the optional parameters below:

SPI_I2S_RDBF_FLAG:	SPI/I ² S receive data buffer full
SPI_I2S_TDBE_FLAG:	SPI/I ² S transmit data buffer empty
I2S_ACS_FLAG:	I2S audio channel state (indicating left/right channel)
I2S_TUERR_FLAG:	I2S transmitter underload error
SPI_CCERR_FLAG:	SPI CRC error
SPI_MMERR_FLAG:	SPI master mode error
SPI_I2S_ROERR_FLAG:	SPI/I ² S receive overflow error
SPI_I2S_BF_FLAG:	SPI/I ² S busy
SPI_CSPAS_FLAG:	SPI CS pulse error flag

Example:

```
/* get receive data buffer full flag */
flag_status status;
status = spi_i2s_flag_get(SPI, SPI_I2S_RDBF_FLAG);
```

5.14.24 spi_i2s_interrupt_flag_get function

The table below describes the function spi_i2s_interrupt_flag_get.

Table 353. spi_i2s_interrupt_flag_get function

Name	Description
Function name	spi_i2s_interrupt_flag_get
Function prototype	flag_status spi_i2s_interrupt_flag_get(spi_type* spi_x, uint32_t spi_i2s_flag);
Function description	Get SPI/I ² S interrupt flag status
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Input parameter 2	spi_i2s_flag : select a flag to clear Refer to the “spi_i2s_flag” description below for details.
Output parameter	NA
Return value	flag_status: Return 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/I ² S receive data buffer full
SPI_I2S_TDBE_FLAG:	SPI/I ² S 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/I ² S receive overflow error
SPI_CSPAS_FLAG:	SPI CS pulse error flag

Example:

```
/* get receive data buffer full flag */  
flag_status status;  
status = spi_i2s_interrupt_flag_get(SPI, SPI_I2S_RDBF_FLAG);
```

5.14.25 spi_i2s_flag_clear function

The table below describes the function spi_i2s_flag_clear.

Table 354. spi_i2s_flag_clear function

Name	Description
Function name	spi_i2s_flag_clear
Function prototype	void spi_i2s_flag_clear(spi_type* spi_x, uint32_t spi_i2s_flag)
Function description	Clear SPI/I ² S flags
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2
Input parameter 2	spi_i2s_flag : select a flag to clear Refer to the “spi_i2s_flag” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

spi_i2s_flag:

SPI/I²S is used for flag selection, including:

SPI_I2S_RDBF_FLAG:	SPI/I ² S receive data buffer full
I2S_TUERR_FLAG:	I2S transmitter underload error
SPI_CCERR_FLAG:	SPI CRC error
SPI_MMERR_FLAG:	SPI master mode error
SPI_I2S_ROERR_FLAG:	SPI/I ² S receive overflow error
SPI_CSPAS_FLAG:	SPI CS pulse error flag

Note: the SPI_I2S_TDBE_FLAG (SPI/I²S transmit data buffer empty), the I2S_ACS_FLAG (Audio channel state) and the 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 (SPI, SPI_I2S_RDBF_FLAG);
```

5.15 SysTick

The SysTick register structure SysTick_Type is defined in the “core_cm0plus.h”:

```
typedef struct
{
    ...

} SysTick_Type;
```

The table below gives a list of the SysTick registers

Table 355. Summary of SysTick registers

Register	Description
ctrl	Controls status register
load	Reload value register
Val	Current counter value register
calib	Calibration register

The table below gives a list of SysTick library functions.

Table 356. 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

5.15.1 systick_clock_source_config function

The table below describes the function systick_clock_source_config.

Table 357. 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);
```

5.15.2 SysTick_Config function

The table below describes the function SysTick_Config

Table 358. SysTick_Config function

Name	Description
Function name	SysTick_Config
Function prototype	uint32_t SysTick_Config(uint32_t ticks);
Function description	Configure SysTick counter reload value and enable interrupt
Input parameter 1	Ticks: SysTick counter interrupt reload value
Output parameter	NA
Return value	Return the setting status of this function, success (0) or failure (1)
Required preconditions	NA
Called functions	NA

Example:

```
/* config systick reload value and enable interrupt */  
SysTick_Config(1000);
```


5.16 TMR

The TMR register structure `tmr_type` is defined in the "at32l021_tmr.h":

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

} tmr_type;
```

The table below gives a list of the TMR registers

Table 359. Summary of TMR registers

Register	Description
ctrl1	TMR control register 1
ctrl2	TMR control register 2
stctrl	TMR slave timer control register
iden	TMR DMA/ interrupt enable register
ists	TMR interrupt status register
swevt	TMR software event register
cm1	TMR channel mode register 1
cm2	TMR channel mode register 2
cctrl	TMR channel control register
cval	TMR counter value register
div	TMR division register
pr	TMR period register
rpr	TMR repetition period channel
c1dt	TMR channel 1 data register
c2dt	TMR channel 2 data register
c3dt	TMR channel 3 data register
c4dt	TMR channel 4 data register
brk	TMR break register
dmactrl	TMR DMA control register
dmadt	TMR DMA data register
rmp	TMR channel input remap register
cm3	TMR channel mode register 3
c5dt	TMR channel 5 data register

The table below gives a list of TMR library functions.

Table 360. Summary of TMR library functions

Function name	Description
tmr_reset	TMR is reset by CRM reset register
tmr_counter_enable	Enable or disable TMR
tmr_output_default_para_init	Initialize TMR output default parameters
tmr_input_default_para_init	Initialize TMR input default parameters
tmr_brkdt_default_para_init	Initialize TMR brkdt default parameters
tmr_base_init	Initialize TMR period and division
tmr_clock_source_div_set	Set TMR clock source frequency division factor
tmr_cnt_dir_set	Set TMR counter direction
tmr_repetition_counter_set	Set repetition period register
tmr_counter_value_set	Set TMR counter value
tmr_counter_value_get	Get TMR counter value
tmr_div_value_set	Set TMR division value
tmr_div_value_get	Get TMR division value
tmr_output_channel_config	Configure TMR output channels
tmr_output_channel_mode_select	Select TMR output channel mode
tmr_period_value_set	Set TMR period value
tmr_period_value_get	Get TMR period value
tmr_channel_value_set	Set TMR channel value
tmr_channel_value_get	Get TMR channel value
tmr_period_buffer_enable	Enable or disable TMR periodic buffer
tmr_output_channel_buffer_enable	Enable or disable TMR output channel buffer
tmr_output_channel_immediately_set	TMR output channel enable immediately
tmr_output_channel_switch_set	Set TMR output channel switch
tmr_one_cycle_mode_enable	Enable or disable TMR one-cycle mode
tmr_overflow_request_source_set	Select TMR overflow event source
tmr_overflow_event_disable	Enable or disable TMR overflow event generation
tmr_channel_enable	Enable or disable TMR channel
tmr_input_channel_filter_set	Set TMR input channel filter
tmr_pwm_input_config	Configure TMR pwm input
tmr_channel1_input_select	Select TMR channel 1 input
tmr_input_channel_divider_set	Set TMR input channel divider
tmr_primary_mode_select	Select TMR master mode
tmr_sub_mode_select	Select TMR slave timer mode
tmr_channel_dma_select	Select TMR channel DMA request source
tmr_hall_select	Select TMR hall mode
tmr_channel_buffer_enable	Enable or disable TMR channel buffer
tmr_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_flag_get	Get TMR flags
tmr_flag_clear	Clear TMR flags

tmr_event_sw_trigger	Software trigger TMR event
tmr_output_enable	Enable or disable TMR output
tmr_internal_clock_set	Set TMR internal clock
tmr_output_channel_polarity_set	Set TMR output channel polarity
tmr_external_clock_config	Set TMR external clock
tmr_external_clock_mode1_config	Set TMR external clock mode 1
tmr_external_clock_mode2_config	Set TMR external clock mode 2
tmr_encoder_mode_config	Set TMR encode mode
tmr_force_output_set	Set TMR forced output
tmr_dma_control_config	Set TMR DMA control
tmr_brkdt_config	Set TMR break mode and dead-time
tmr_iremap_config	Set TMR internal remap

5.16.1 tmr_reset function

The table below describes the function tmr_reset.

Table 361. tmr_reset function

Name	Description
Function name	tmr_reset
Function prototype	void tmr_reset(tmr_type *tmr_x);
Function description	TMR is reset by CRM reset register.
Input parameter	tmr_x: select TMR peripheral, including: TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

Example:

```
tmr_reset(TMR1);
```

5.16.2 tmr_counter_enable function

The table below describes the function tmr_counter_enable.

Table 362. tmr_counter_enable function

Name	Description
Function name	tmr_counter_enable
Function prototype	void tmr_counter_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR
Input parameter 1	tmr_x: select TMR peripheral, including: TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Input parameter 2	new_state: indicates counter status, ON (TRUE) or OFF (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_counter_enable(TMR1, TRUE);
```

5.16.3 tmr_output_default_para_init function

The table below describes the function `tmr_output_default_para_init`.

Table 363. tmr_output_default_para_init function

Name	Description
Function name	<code>tmr_output_default_para_init</code>
Function prototype	<code>void tmr_output_default_para_init(tmr_output_config_type *tmr_output_struct);</code>
Function description	Initialize tmr output default parameters
Input parameter	<code>tmr_output_struct</code> : <code>tmr_output_config_type</code> pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of members of the function `tmr_output_struct`.

Table 364. tmr_output_struct default values

Member	Default values
<code>oc_mode</code>	<code>TMR_OUTPUT_CONTROL_OFF</code>
<code>oc_idle_state</code>	<code>FALSE</code>
<code>occ_idle_state</code>	<code>FALSE</code>
<code>oc_polarity</code>	<code>TMR_OUTPUT_ACTIVE_HIGH</code>
<code>occ_polarity</code>	<code>TMR_OUTPUT_ACTIVE_HIGH</code>
<code>oc_output_state</code>	<code>FALSE</code>
<code>occ_output_state</code>	<code>FALSE</code>

Example:

```
tmr_output_config_type tmr_output_struct;
tmr_output_default_para_init(&tmr_output_struct);
```

5.16.4 tmr_input_default_para_init function

The table below describes the function `tmr_input_default_para_init`.

Table 365. tmr_input_default_para_init function

Name	Description
Function name	<code>tmr_input_default_para_init</code>
Function prototype	<code>void tmr_input_default_para_init(tmr_input_config_type *tmr_input_struct);</code>
Function description	Initialize TMR input default parameters
Input parameter	<code>tmr_input_struct</code> : <code>tmr_input_config_type</code> pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of members of the function `tmr_input_struct`.

Table 366. 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);
```

5.16.5 tmr_brkdt_default_para_init function

The table below describes the function tmr_brkdt_default_para_init.

Table 367. tmr_brkdt_default_para_init function

Name	Description
Function name	tmr_brkdt_default_para_init
Function prototype	void tmr_brkdt_default_para_init(tmr_brkdt_config_type *tmr_brkdt_struct);
Function description	Initialize TMR brkdt default parameters
Input parameter	tmr_brkdt_struct: tmr_brkdt_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of members of the function tmr_brkdt_struct.

Table 368. tmr_brkdt_struct default values

Member	Default values
brk_filter_value	0x0
deadtime	0x0
brk_polarity	TMR_BRK_INPUT_ACTIVE_LOW
wp_level	TMR_WP_OFF
auto_output_enable	FALSE
fcsoen_state	FALSE
fcsodis_state	FALSE
brk_enable	FALSE

Example:

```
tmr_brkdt_config_type tmr_brkdt_struct;
tmr_brkdt_default_para_init(&tmr_brkdt_struct);
```

5.16.6 tmr_base_init function

The table below describes the function tmr_base_init.

Table 369. tmr_base_init function

Name	Description
Function name	tmr_base_init
Function prototype	void tmr_base_init(tmr_type* tmr_x, uint32_t tmr_pr, uint32_t tmr_div);
Function description	Initialize TMR period and division
Input parameter 1	tmr_x: TMR peripheral including: TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Input parameter 2	tmr_pr: timer period value, 0x0000~0xFFFF for 16-bit timer, and 0x0000_0000~0xFFFF_FFFF for 32-bit timer,
Input parameter 3	tmr_div: timer division value, 0x0000~0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_base_init(TMR1, 0xFFFF, 0xFFFF);
```

5.16.7 tmr_clock_source_div_set function

The table below describes the function tmr_clock_source_div_set.

Table 370. tmr_clock_source_div_set function

Name	Description
Function name	tmr_clock_source_div_set
Function prototype	void tmr_clock_source_div_set(tmr_type *tmr_x, tmr_clock_division_type tmr_clock_div);
Function description	Set TMR clock source division
Input parameter 1	tmr_x: TMR peripheral, including: TMR1, TMR3, TMR14, TMR15, TMR16, TMR17
Input parameter 2	tmr_clock_div: timer clock source frequency division factor
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_clock_div

Select TMR clock source frequency division factor

TMR_CLOCK_DIV1: Divided by 1

TMR_CLOCK_DIV2: Divided by 2

TMR_CLOCK_DIV4: Divided by 4

Example:

```
tmr_clock_source_div_set(TMR1, TMR_CLOCK_DIV4);
```

5.16.8 tmr_cnt_dir_set function

The table below describes the function tmr_cnt_dir_set.

Table 371. tmr_cnt_dir_set function

Name	Description
Function name	tmr_cnt_dir_set
Function prototype	void tmr_cnt_dir_set(tmr_type *tmr_x, tmr_count_mode_type tmr_cnt_dir);
Function description	Set TMR counter direction
Input parameter 1	tmr_x: indicates the selected TMR peripheral, including: TMR1, TMR3
Input parameter 2	tmr_cnt_dir: timer counting direction
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_cnt_dir

Select timer counting direction.

TMR_COUNT_UP: Up counting
 TMR_COUNT_DOWN: Down counting
 TMR_COUNT_TWO_WAY_1: Center-aligned mode (up/down counting) 1
 TMR_COUNT_TWO_WAY_2: Center-aligned mode (up/down counting) 2
 TMR_COUNT_TWO_WAY_3: Center-aligned mode (up/down counting) 3

Example:

```
tmr_cnt_dir_set(TMR1, TMR_COUNT_UP);
```

5.16.9 tmr_repetition_counter_set function

The table below describes the function tmr_repetition_counter_set.

Table 372. tmr_repetition_counter_set function

Name	Description
Function name	tmr_repetition_counter_set
Function prototype	void tmr_repetition_counter_set(tmr_type *tmr_x, uint16_t tmr_rpr_value);
Function description	Set repetition period register (rpr)
Input parameter 1	tmr_x: TMR peripheral, it includes: TMR1, TMR15, TMR16, TMR17
Input parameter 2	tmr_rpr_value: timer repetition period value, it can be 0x00~0xFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_repetition_counter_set(TMR1, 0x10);
```

5.16.10 tmr_counter_value_set function

The table below describes the function tmr_counter_value_set.

Table 373. tmr_counter_value_set function

Name	Description
Function name	tmr_counter_value_set
Function prototype	void tmr_counter_value_set(tmr_type *tmr_x, uint32_t tmr_cnt_value);
Function description	Set TMR counter value
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Input parameter 2	tmr_cnt_value: timer counter value, 0x0000~0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_counter_value_set(TMR1, 0xFFFF);
```

5.16.11 tmr_counter_value_get function

The table below describes the function tmr_counter_value_get.

Table 374. tmr_counter_value_get function

Name	Description
Function name	tmr_counter_value_get
Function prototype	uint32_t tmr_counter_value_get(tmr_type *tmr_x);
Function description	Get TMR counter value
Input parameter	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Output parameter	NA
Return value	Timer counter value
Required preconditions	NA
Called functions	NA

Example:

```
uint32_t counter_value;
counter_value = tmr_counter_value_get(TMR1);
```


5.16.12 tmr_div_value_set function

The table below describes the function tmr_div_value_set.

Table 375. tmr_div_value_set function

Name	Description
Function name	tmr_div_value_set
Function prototype	void tmr_div_value_set(tmr_type *tmr_x, uint32_t tmr_div_value);
Function description	Set TMR frequency division value
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Input parameter 2	tmr_div_value: timer frequency division value. 0x0000~0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_div_value_set(TMR1, 0xFFFF);
```

5.16.13 tmr_div_value_get function

The table below describes the function tmr_div_value_get.

Table 376. tmr_div_value_get function

Name	Description
Function name	tmr_div_value_get
Function prototype	uint32_t tmr_div_value_get(tmr_type *tmr_x);
Function description	Get TMR frequency division value
Input parameter	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Output parameter	NA
Return value	Timer frequency division value
Required preconditions	NA
Called functions	NA

Example:

```
uint32_t div_value;
div_value = tmr_div_value_get(TMR1);
```

5.16.14 tmr_output_channel_config function

The table below describes the function tmr_output_channel_config.

Table 377. tmr_output_channel_config function

Name	Description
Function name	tmr_output_channel_config
Function prototype	void tmr_output_channel_config(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, tmr_output_config_type *tmr_output_struct);
Function description	Configure TMR output channels
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR14, TMR15, TMR16, TMR17
Input parameter 2	tmr_channel: timer channel
Input parameter 3	tmr_output_struct: tmr_output_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel.

TMR_SELECT_CHANNEL_1: Channel 1

TMR_SELECT_CHANNEL_2: Channel 2

TMR_SELECT_CHANNEL_3: Channel 3

TMR_SELECT_CHANNEL_4: Channel 4

TMR_SELECT_CHANNEL_5: Channel 5

tmr_output_config_type structure

tmr_output_config_type is defined in the at32l021_tmr.h:

typedef struct

```
{
    tmr_output_control_mode_type    oc_mode;
    confirm_state                   oc_idle_state;
    confirm_state                   occ_idle_state;
    tmr_output_polarity_type        oc_polarity;
    tmr_output_polarity_type        occ_polarity;
    confirm_state                   oc_output_state;
    confirm_state                   occ_output_state;
} tmr_output_config_type;
```

oc_mode

Set output channel mode, that is, to configure channel original signals (CxORAW).

TMR_OUTPUT_CONTROL_OFF: Disconnect channel output (CxOUT) from CxORAW

TMR_OUTPUT_CONTROL_HIGH: CxORAW high

TMR_OUTPUT_CONTROL_LOW: CxORAW low

TMR_OUTPUT_CONTROL_SWITCH: Switch CxORAW level

TMR_OUTPUT_CONTROL_FORCE_LOW: CxORAW forced low

TMR_OUTPUT_CONTROL_FORCE_HIGH: CxORAW forced high

TMR_OUTPUT_CONTROL_PWM_MODE_A: PWM A mode

TMR_OUTPUT_CONTROL_PWM_MODE_B: PWM B mode

oc_idle_state

Set output channel idle state.

FALSE: Output channel idle state is 0

TRUE: Output channel idle state is 1

occ_idle_state

Set complementary output channel idle state.

FALSE: Complementary output channel idle state is 0

TRUE: Complementary output channel idle state is 1

oc_polarity

Set the polarity of output channels.

TMR_OUTPUT_ACTIVE_HIGH: Active high

TMR_OUTPUT_ACTIVE_LOW: Active low

occ_polarity

Set the polarity of complementary output channels.

TMR_OUTPUT_ACTIVE_HIGH: Active high

TMR_OUTPUT_ACTIVE_LOW: Active low

oc_output_state

Set the state of output channels.

FALSE: Output channel OFF

TRUE: Output channel ON

occ_output_state

Set the state of complementary output channels.

FALSE: Complementary output channel OFF

TRUE: Complementary output channel ON

Example:

```
tmr_output_config_type tmr_output_struct;  
tmr_output_struct.oc_mode = TMR_OUTPUT_CONTROL_OFF;  
tmr_output_struct.oc_output_state = TRUE;  
tmr_output_struct.oc_polarity = TMR_OUTPUT_ACTIVE_HIGH;  
tmr_output_struct.oc_idle_state = TRUE;  
tmr_output_struct.occ_output_state = TRUE;  
tmr_output_struct.occ_polarity = TMR_OUTPUT_ACTIVE_HIGH;  
tmr_output_struct.occ_idle_state = TRUE;  
tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_1, &tmr_output_struct);
```

5.16.15 tmr_output_channel_mode_select function

The table below describes the function `tmr_output_channel_mode_select`.

Table 378. tmr_output_channel_mode_select function

Name	Description
Function name	<code>tmr_output_channel_mode_select</code>
Function prototype	<code>void tmr_output_channel_mode_select(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, tmr_output_control_mode_type oc_mode);</code>
Function description	Select TMR output channel mode
Input parameter 1	<code>tmr_x</code> : indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR14, TMR15, TMR16, TMR17
Input parameter 2	<code>tmr_channel</code> : refer to the “ tmr_channel ” descriptions below for details
Input parameter 3	<code>oc_mode</code> : refer to the “ oc_mode ” descriptions below for details
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel.

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

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) from CxORAW

TMR_OUTPUT_CONTROL_HIGH: CxORAW high

TMR_OUTPUT_CONTROL_LOW: CxORAW low

TMR_OUTPUT_CONTROL_SWITCH: Switch CxORAW level

TMR_OUTPUT_CONTROL_FORCE_LOW: CxORAW forced low

TMR_OUTPUT_CONTROL_FORCE_HIGH: CxORAW forced high

TMR_OUTPUT_CONTROL_PWM_MODE_A: PWM A mode

TMR_OUTPUT_CONTROL_PWM_MODE_B: PWM B mode

Example:

```
tmr_output_channel_mode_select(TMR1, TMR_SELECT_CHANNEL_1, TMR_OUTPUT_CONTROL_SWITCH);
```

5.16.16 tmr_period_value_set function

The table below describes the function tmr_period_value_set.

Table 379. tmr_period_value_set function

Name	Description
Function name	tmr_period_value_set
Function prototype	void tmr_period_value_set(tmr_type *tmr_x, uint32_t tmr_pr_value);
Function description	Set TMR period value
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Input parameter 2	tmr_pr_value: timer period value., 0x0000~0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_period_value_set(TMR1, 0xFFFF);
```

5.16.17 tmr_period_value_get function

The table below describes the function tmr_period_value_get.

Table 380. tmr_period_value_get function

Name	Description
Function name	tmr_period_value_get
Function prototype	uint32_t tmr_period_value_get(tmr_type *tmr_x);
Function description	Get TMR period value
Input parameter	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Output parameter	NA
Return value	Timer period value
Required preconditions	NA
Called functions	NA

Example:

```
uint32_t pr_value;
pr_value = tmr_period_value_get(TMR1);
```

5.16.18 tmr_channel_value_set function

The table below describes the function tmr_channel_value_set.

Table 381. tmr_channel_value_set function

Name	Description
Function name	tmr_channel_value_set
Function prototype	void tmr_channel_value_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, uint32_t tmr_channel_value);
Function description	Set TMR channel value
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR14, TMR15, TMR16, TMR17
Input parameter 2	tmr_channel: timer channel
Input parameter 3	tmr_channel_value: timer channel value. 0x0000~0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel.

TMR_SELECT_CHANNEL_1:	Channel 1
TMR_SELECT_CHANNEL_2:	Channel 2
TMR_SELECT_CHANNEL_3:	Channel 3
TMR_SELECT_CHANNEL_4:	Channel 4
TMR_SELECT_CHANNEL_5:	Channel 5

Example:

```
tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_1, 0xFFFF);
```

5.16.19 tmr_channel_value_get function

The table below describes the function tmr_channel_value_get.

Table 382. tmr_channel_value_get function

Name	Description
Function name	tmr_channel_value_get
Function prototype	uint32_t tmr_channel_value_get(tmr_type *tmr_x, tmr_channel_select_type tmr_channel);
Function description	Get TMR channel value
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR14, TMR15, TMR16, TMR17
Input parameter 2	tmr_channel: timer channel
Output parameter	Timer channel value
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel.

TMR_SELECT_CHANNEL_1: TMR channel 1

TMR_SELECT_CHANNEL_2: TMR channel 2

TMR_SELECT_CHANNEL_3: TMR channel 3

TMR_SELECT_CHANNEL_4: TMR channel 4

TMR_SELECT_CHANNEL_5: TMR channel 5

Example:

```
uint32_t ch_value;
ch_value = tmr_channel_value_get(TMR1, TMR_SELECT_CHANNEL_1);
```

5.16.20 tmr_period_buffer_enable function

The table below describes the function tmr_period_buffer_enable.

Table 383. tmr_period_buffer_enable function

Name	Description
Function name	tmr_period_buffer_enable
Function prototype	void tmr_period_buffer_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR period buffer
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Input parameter 2	new_state: indicates the status of period buffer. It can be Enable (TRUE) or Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_period_buffer_enable(TMR1, TRUE);
```

5.16.21 tmr_output_channel_buffer_enable function

The table below describes the function tmr_output_channel_buffer_enable.

Table 384. tmr_output_channel_buffer_enable function

Name	Description
Function name	tmr_output_channel_buffer_enable
Function prototype	void tmr_output_channel_buffer_enable(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, confirm_state new_state);
Function description	Enable or disable TMR output channel buffer
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR14, TMR15, TMR16, TMR17
Input parameter 2	tmr_channel: timer channel
Input parameter 3	new_state: indicates the status of output channel buffer. It can be Enable (TRUE) or Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel.

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

TMR_SELECT_CHANNEL_5: Timer channel 5

Example:

```
tmr_output_channel_buffer_enable(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```


5.16.22 tmr_output_channel_immediately_set function

The table below describes the function `tmr_output_channel_immediately_set`.

Table 385. tmr_output_channel_immediately_set function

Name	Description
Function name	<code>tmr_output_channel_immediately_set</code>
Function prototype	<code>void tmr_output_channel_immediately_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, confirm_state new_state);</code>
Function description	Enable TMR output channel immediately
Input parameter 1	<code>tmr_x</code> : indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR14, TMR15, TMR16, TMR17
Input parameter 2	<code>tmr_channel</code> : timer channel
Input parameter 3	<code>new_state</code> : indicates the status of output channel enable. This parameter can be Enable (TRUE) or Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel.

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

TMR_SELECT_CHANNEL_5: Timer channel 5

Example:

```
tmr_output_channel_immediately_set(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

5.16.23 tmr_output_channel_switch_set function

The table below describes the function `tmr_output_channel_switch_set`.

Table 386. tmr_output_channel_switch_set function

Name	Description
Function name	<code>tmr_output_channel_switch_set</code>
Function prototype	<code>void tmr_output_channel_switch_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, confirm_state new_state);</code>
Function description	Set TMR output channel switch
Input parameter 1	<code>tmr_x</code> : indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR14, TMR15, TMR16, TMR17
Input parameter 2	<code>tmr_channel</code> : timer channel
Input parameter 3	<code>new_state</code> : indicates the status of output channel switch. This parameter can be Enable (TRUE) or Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel.

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

TMR_SELECT_CHANNEL_5: Timer channel 5

Example:

```
tmr_output_channel_switch_set(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

5.16.24 tmr_one_cycle_mode_enable function

The table below describes the function `tmr_one_cycle_mode_enable`.

Table 387. tmr_one_cycle_mode_enable function

Name	Description
Function name	<code>tmr_one_cycle_mode_enable</code>
Function prototype	<code>void tmr_one_cycle_mode_enable(tmr_type *tmr_x, confirm_state new_state);</code>
Function description	Enable or disable TMR one-cycle mode
Input parameter 1	<code>tmr_x</code> : indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Input parameter 2	<code>new_state</code> : indicates the status of one-cycle mode. This parameter can be Enable (TRUE) or Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_one_cycle_mode_enable(TMR1, TRUE);
```

5.16.25 tmr_overflow_request_source_set function

The table below describes the function tmr_overflow_request_source_set.

Table 388. tmr_overflow_request_source_set function

Name	Description
Function name	tmr_overflow_request_source_set
Function prototype	void tmr_overflow_request_source_set(tmr_type *tmr_x, confirm_state new_state);
Function description	Select TMR overflow event sources
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Input parameter 2	new_state: indicates the overflow event source.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

Select an overflow event source.

FALSE: Counter overflow, OVFSWTR being set, overflow event from slave mode timer controller

TRUE: Counter overflow only.

Example:

```
tmr_overflow_request_source_set(TMR1, TRUE);
```

5.16.26 tmr_overflow_event_disable function

The table below describes the function tmr_overflow_event_disable.

Table 389. tmr_overflow_event_disable function

Name	Description
Function name	tmr_overflow_event_disable
Function prototype	void tmr_overflow_event_disable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR overflow event generation
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Input parameter 2	new_state: indicates the status of overflow event generation.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

new_state

Select the status of overflow event generation.

FALSE: Enable overflow event generation, which can be generated from the following:

- Counter overflow
- Set OVFSWTR=1
- Overflow event from slave mode timer controller

TRUE: Disable overflow event generation

Example:

```
tmr_overflow_event_disable(TMR1, TRUE);
```

5.16.27 tmr_input_channel_init function

The table below describes the function tmr_input_channel_init.

Table 390. tmr_input_channel_init function

Name	Description
Function name	tmr_input_channel_init
Function prototype	void tmr_input_channel_init(tmr_type *tmr_x, tmr_input_config_type *input_struct, tmr_channel_input_divider_type divider_factor);
Function description	Initialize TMR input channels
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR14, TMR15, TMR16, TMR17
Input parameter 2	input_struct: tmr_input_config_type pointer
Input parameter 3	divider_factor: input channel frequency division factor
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_input_config_type structure

tmr_input_config_type is defined in the at32l021_tmr.h:

typedef struct

```
{
    tmr_channel_select_type      input_channel_select;
    tmr_input_polarity_type      input_polarity_select;
    tmr_input_direction_mapped_type  input_mapped_select;
    uint8_t                     input_filter_value;
} tmr_input_config_type;
```

input_channel_select

Select a TMR input channel.

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

input_polarity_select

Select the polarity of input channels.

TMR_INPUT_RISING_EDGE: Rising edge

TMR_INPUT_FALLING_EDGE: Falling edge

TMR_INPUT_BOTH_EDGE: Both edges (Rising edge and Falling edge)

input_mapped_select

Select input channel mapping.

TMR_CC_CHANNEL_MAPPED_DIRECT:

TMR input channel 1,2,3 and 4 is linked to C1IRAW, C2IRAW, C3IRAW and C4IRAW respectively.

TMR_CC_CHANNEL_MAPPED_INDIRECT:

TMR input channel 1,2,3 and 4 is linked to C2IRAW, C1IRAW, C4IRAW and C3IRAW respectively.

TMR_CC_CHANNEL_MAPPED_STI:

TMR input channel is mapped on STI

input_filter_value

Select an input channel filter value, between 0x00~0x0F

divider_factor

Select input channel frequency division factor.

TMR_CHANNEL_INPUT_DIV_1: Divided by 1

TMR_CHANNEL_INPUT_DIV_2: Divided by 2

TMR_CHANNEL_INPUT_DIV_4: Divided by 4

TMR_CHANNEL_INPUT_DIV_8: Divided by 8

Example:

```
tmr_input_config_type tmr_input_config_struct;
tmr_input_config_struct.input_channel_select = TMR_SELECT_CHANNEL_2;
tmr_input_config_struct.input_mapped_select = TMR_CC_CHANNEL_MAPPED_DIRECT;
tmr_input_config_struct.input_polarity_select = TMR_INPUT_RISING_EDGE;
tmr_input_config_struct.input_filter_value = 0x00;
tmr_input_channel_init(TMR1, &tmr_input_config_struct, TMR_CHANNEL_INPUT_DIV_1);
```

5.16.28 tmr_channel_enable function

The table below describes the function tmr_channel_enable.

Table 391. tmr_channel_enable function

Name	Description
Function name	tmr_channel_enable
Function prototype	void tmr_channel_enable(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, confirm_state new_state);
Function description	Enable or disable TMR channels
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR14, TMR15, TMR16, TMR17
Input parameter 2	tmr_channel: timer channel
Input parameter 3	new_state: indicates the status of timer channels. This parameter can be Enable (TRUE) or Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel.

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_1C: Complementary channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_2C: Complementary channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_3C: Complementary channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

Example:

```
tmr_channel_enable(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

5.16.29 tmr_input_channel_filter_set function

The table below describes the function `tmr_input_channel_filter_set`.

Table 392. tmr_input_channel_filter_set function

Name	Description
Function name	<code>tmr_input_channel_filter_set</code>
Function prototype	<code>void tmr_input_channel_filter_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, uint16_t filter_value);</code>
Function description	Set TMR input channel filter
Input parameter 1	<code>tmr_x</code> : indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR14, TMR15, TMR16, TMR17
Input parameter 2	<code>tmr_channel</code> : timer channel
Input parameter 3	<code>filter_value</code> : set channel filter value, 0x00~0x0F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel.

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

Example:

```
tmr_input_channel_filter_set(TMR1, TMR_SELECT_CHANNEL_1, 0x0F);
```

5.16.30 tmr_pwm_input_config function

The table below describes the function `tmr_pwm_input_config`.

Table 393. tmr_pwm_input_config function

Name	Description
Function name	<code>tmr_pwm_input_config</code>
Function prototype	<code>void tmr_pwm_input_config(tmr_type *tmr_x, tmr_input_config_type *input_struct, tmr_channel_input_divider_type divider_factor);</code>
Function description	Configure TMR pwm input
Input parameter 1	<code>tmr_x</code> : indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR15
Input parameter 2	<code>input_struct</code> : <code>tmr_input_config_type</code> pointer
Input parameter 3	<code>divider_factor</code> : input channel frequency division factor
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

input_struct

Point to the `tmr_input_config_type`, see [tmr_input_config_type](#) for details.

divider_factor

Select input channel frequency division factor

TMR_CHANNEL_INPUT_DIV_1: Divided by 1

TMR_CHANNEL_INPUT_DIV_2: Divided by 2

TMR_CHANNEL_INPUT_DIV_4: Divided by 4

TMR_CHANNEL_INPUT_DIV_8: Divided by 8

Example:

```
tmr_input_config_type tmr_ic_init_structure;
tmr_ic_init_structure.input_filter_value = 0;
tmr_ic_init_structure.input_channel_select = TMR_SELECT_CHANNEL_2;
tmr_ic_init_structure.input_mapped_select = TMR_CC_CHANNEL_MAPPED_DIRECT;
tmr_ic_init_structure.input_polarity_select = TMR_INPUT_RISING_EDGE;
tmr_pwm_input_config(TMR1, &tmr_ic_init_structure, TMR_CHANNEL_INPUT_DIV_1);
```

5.16.31 tmr_channel1_input_select function

The table below describes the function tmr_channel1_input_select.

Table 394. tmr_channel1_input_select function

Name	Description
Function name	tmr_channel1_input_select
Function prototype	void tmr_channel1_input_select(tmr_type *tmr_x, tmr_channel1_input_connected_type ch1_connect);
Function description	Select TMR channel 1 input
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3
Input parameter 2	ch1_connect: channel 1 input selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

ch1_connect

Select channel 1 input.

TMR_CHANNEL1_CONNECTED_C1IRAW: CH1 pin is connected to C1IRAW

TMR_CHANNEL1_2_3_CONNECTED_C1IRAW_XOR: Connect the XOR results of CH1, CH2 and CH3 pins to C1IRAW

Example:

```
tmr_channel1_input_select(TMR1, TMR_CHANNEL1_2_3_CONNECTED_C1IRAW_XOR);
```

5.16.32 tmr_input_channel_divider_set function

The table below describes the function tmr_input_channel_divider_set.

Table 395. tmr_input_channel_divider_set function

Name	Description
Function name	tmr_input_channel_divider_set
Function prototype	void tmr_input_channel_divider_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, tmr_channel_input_divider_type divider_factor);
Function description	Set TMR input channel divider
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR14, TMR15, TMR16, TMR17
Input parameter 2	tmr_channel: timer channel
Input parameter 3	divider_factor: input channel frequency division factor
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel.

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

divider_factor

Select input channel frequency division factor

TMR_CHANNEL_INPUT_DIV_1: Divided by 1

TMR_CHANNEL_INPUT_DIV_2: Divided by 2

TMR_CHANNEL_INPUT_DIV_4: Divided by 4

TMR_CHANNEL_INPUT_DIV_8: Divided by 8

Example:

```
tmr_input_channel_divider_set(TMR1, TMR_SELECT_CHANNEL_1, TMR_CHANNEL_INPUT_DIV_2);
```


5.16.33 tmr_primary_mode_select function

The table below describes the function tmr_primary_mode_select.

Table 396. tmr_primary_mode_select function

Name	Description
Function name	tmr_primary_mode_select
Function prototype	void tmr_primary_mode_select(tmr_type *tmr_x, tmr_primary_select_type primary_mode);
Function description	Select TMR primary (master) mode
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6
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);
```

5.16.34 tmr_sub_mode_select function

The table below describes the function tmr_sub_mode_select.

Table 397. tmr_sub_mode_select function

Name	Description
Function name	tmr_sub_mode_select
Function prototype	void tmr_sub_mode_select(tmr_type *tmr_x, tmr_sub_mode_select_type sub_mode);
Function description	Select TMR slave timer mode
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR15
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_DISABLE:	Disable
TMR_SUB_ENCODER_MODE_A:	Encoder mode A
TMR_SUB_ENCODER_MODE_B:	Encoder mode B
TMR_SUB_ENCODER_MODE_C:	Encoder mode C
TMR_SUB_RESET_MODE:	Reset
TMR_SUB_HANG_MODE:	Suspend
TMR_SUB_TRIGGER_MODE:	Trigger
TMR_SUB_EXTERNAL_CLOCK_MODE_A:	External clock A

Example:

```
tmr_sub_mode_select(TMR1, TMR_SUB_HANG_MODE);
```

5.16.35 tmr_channel_dma_select function

The table below describes the function tmr_channel_dma_select.

Table 398. tmr_channel_dma_select function

Name	Description
Function name	tmr_channel_dma_select
Function prototype	void tmr_channel_dma_select(tmr_type *tmr_x, tmr_dma_request_source_type cc_dma_select);
Function description	Select TMR channel DMA request source
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR15, TMR16, TMR17
Input parameter 2	cc_dma_select: TMR channel DMA request source
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

cc_dma_select

Select DMA request source for TMR channels.

TMR_DMA_REQUEST_BY_CHANNEL: DMA request upon a channel event (CxIF = 1)

TMR_DMA_REQUEST_BY_OVERFLOW: DMA request upon an overflow event (OVFIF = 1)

Example:

```
tmr_channel_dma_select(TMR1, TMR_DMA_REQUEST_BY_OVERFLOW);
```

5.16.36 tmr_hall_select function

The table below describes the function tmr_hall_select

Table 399. tmr_hall_select function

Name	Description
Function name	tmr_hall_select
Function prototype	void tmr_hall_select(tmr_type *tmr_x, confirm_state new_state);
Function description	Select TMR hall mode
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR15, TMR16, TMR17
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);
```

5.16.37 tmr_channel_buffer_enable function

The table below describes the function tmr_channel_buffer_enable.

Table 400. tmr_channel_buffer_enable function

Name	Description
Function name	tmr_channel_buffer_enable
Function prototype	void tmr_channel_buffer_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR channel buffer
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR15, TMR16, TMR17
Input parameter 2	new_state: indicates the status of TMR channel buffer. This parameter can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_channel_buffer_enable(TMR1, TRUE);
```

5.16.38 tmr_trgout2_enable function

The table below describes the function tmr_trgout2_enable.

Table 401. 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: indicates the selected TMR peripheral, it can be TMR1
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);
```

5.16.39 tmr_trigger_input_select function

The table below describes the function tmr_trigger_input_select.

Table 402. tmr_trigger_input_select function

Name	Description
Function name	tmr_trigger_input_select
Function prototype	void tmr_trigger_input_select(tmr_type *tmr_x, sub_tmr_input_sel_type trigger_select);
Function description	Select TMR slave timer trigger input
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR15
Input parameter 2	trigger_select: select TMR slave timer trigger input
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

trigger_select

Select TMR slave timer trigger input.

TMR_SUB_INPUT_SEL_IS0: Internal input 0
TMR_SUB_INPUT_SEL_IS1: Internal input 1
TMR_SUB_INPUT_SEL_IS2: Internal input 2
TMR_SUB_INPUT_SEL_IS3: Internal input 3
TMR_SUB_INPUT_SEL_C1INC: C1IRAW input detection
TMR_SUB_INPUT_SEL_C1DF1: Filter input channel 1
TMR_SUB_INPUT_SEL_C2DF2: Filter input channel 2
TMR_SUB_INPUT_SEL_EXTIN: External input channel EXT

Example:

```
tmr_trigger_input_select(TMR1, TMR_SUB_INPUT_SEL_IS0);
```

5.16.40 tmr_sub_sync_mode_set function

The table below describes the function tmr_sub_sync_mode_set.

Table 403. tmr_sub_sync_mode_set function

Name	Description
Function name	tmr_sub_sync_mode_set
Function prototype	void tmr_sub_sync_mode_set(tmr_type *tmr_x, confirm_state new_state);
Function description	Set TMR slave timer synchronization mode
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR15
Input parameter 2	new_state: indicates the status of TMR slave timer synchronization mode This parameter can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_sub_sync_mode_set(TMR1, TRUE);
```

5.16.41 tmr_dma_request_enable function

The table below describes the function tmr_dma_request_enable.

Table 404. tmr_dma_request_enable function

Name	Description
Function name	tmr_dma_request_enable
Function prototype	void tmr_dma_request_enable(tmr_type *tmr_x, tmr_dma_request_type dma_request, confirm_state new_state);
Function description	Enable or disable TMR DMA request
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Input parameter 2	dma_request: DMA request
Input parameter 3	new_state: indicates the status of DMA request. This parameter can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dma_request

Select a DMA request.

TMR_OVERFLOW_DMA_REQUEST: Overflow event DMA request
 TMR_C1_DMA_REQUEST: Channel 1 DMA request
 TMR_C2_DMA_REQUEST: Channel 2 DMA request
 TMR_C3_DMA_REQUEST: Channel 3 DMA request
 TMR_C4_DMA_REQUEST: Channel 4 DMA request
 TMR_HALL_DMA_REQUEST: HALL event DMA request
 TMR_TRIGGER_DMA_REQUEST: Trigger event DMA request

Example:

```
tmr_dma_request_enable(TMR1, TMR_OVERFLOW_DMA_REQUEST, TRUE);
```

5.16.42 tmr_interrupt_enable function

The table below describes the function tmr_interrupt_enable.

Table 405. tmr_interrupt_enable function

Name	Description
Function name	tmr_interrupt_enable
Function prototype	void tmr_interrupt_enable(tmr_type *tmr_x, uint32_t tmr_interrupt, confirm_state new_state);
Function description	Enable or disable TMR interrupts
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Input parameter 2	tmr_interrupt: TMR interrupts
Input parameter 3	new_state: indicates the status of TMR interrupts. This parameter can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_interrupt

Select a TMR interrupt.

TMR_OVF_INT: Overflow event interrupt
 TMR_C1_INT: Channel 1 event interrupt
 TMR_C2_INT: Channel 2 event interrupt
 TMR_C3_INT: Channel 3 event interrupt
 TMR_C4_INT: Channel 4 event interrupt
 TMR_HALL_INT: HALL event interrupt
 TMR_TRIGGER_INT: Trigger event interrupt
 TMR_BRK_INT: Break event interrupt

Example:

```
tmr_interrupt_enable(TMR1, TMR_OVF_INT, TRUE);
```

5.16.43 tmr_interrupt_flag_get function

The table below describes the function tmr_interrupt_flag_get.

Table 406. 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 interrupt flag status
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Input parameter 2	tmr_flag: Flag selection Refer to the “tmr_flag” description below for details.
Output parameter	NA
Return value	flag_status: Return SET or RESET
Required preconditions	NA
Called functions	NA

tmr_flag

This is used for flag selection, including:

TMR_OVF_FLAG:	Overflow interrupt flag
TMR_C1_FLAG:	Channel 1 interrupt flag
TMR_C2_FLAG:	Channel 2 interrupt flag
TMR_C3_FLAG:	Channel 3 interrupt flag
TMR_C4_FLAG:	Channel 4 interrupt flag
TMR_HALL_FLAG:	HALL interrupt flag
TMR_TRIGGER_FLAG:	Trigger interrupt flag
TMR_BRK_FLAG:	Break interrupt flag

Example:

```
if(tmr_interrupt_flag_get (TMR1, TMR_OVF_FLAG) != RESET)
```


5.16.44 tmr_flag_get function

The table below describes the function `tmr_flag_get`.

Table 407. tmr_flag_get function

Name	Description
Function name	<code>tmr_flag_get</code>
Function prototype	<code>flag_status_tmr_flag_get(tmr_type *tmr_x, uint32_t tmr_flag);</code>
Function description	Get flag status
Input parameter 1	<code>tmr_x</code> : indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Input parameter 2	<code>tmr_flag</code> : Flag selection Refer to the “tmr_flag” description below for details.
Output parameter	NA
Return value	<code>flag_status</code> : indicates the status of flags Return SET or RESET
Required preconditions	NA
Called functions	NA

tmr_flag

This is used for flag selection, including:

<code>TMR_OVF_FLAG</code> :	Overflow interrupt flag
<code>TMR_C1_FLAG</code> :	Channel 1 interrupt flag
<code>TMR_C2_FLAG</code> :	Channel 2 interrupt flag
<code>TMR_C3_FLAG</code> :	Channel 3 interrupt flag
<code>TMR_C4_FLAG</code> :	Channel 4 interrupt flag
<code>TMR_C5_FLAG</code> :	Channel 5 interrupt flag
<code>TMR_HALL_FLAG</code> :	HALL interrupt flag
<code>TMR_TRIGGER_FLAG</code> :	Trigger interrupt flag
<code>TMR_BRK_FLAG</code> :	Break interrupt flag
<code>TMR_C1_RECAPTURE_FLAG</code> :	Channel 1 recapture flag
<code>TMR_C2_RECAPTURE_FLAG</code> :	Channel 2 recapture flag
<code>TMR_C3_RECAPTURE_FLAG</code> :	Channel 3 recapture flag
<code>TMR_C4_RECAPTURE_FLAG</code> :	Channel 4 recapture flag

Example:

```
if(tmr_flag_get(TMR1, TMR_OVF_FLAG) != RESET)
```

5.16.45 tmr_flag_clear function

The table below describes the function `tmr_flag_clear`.

Table 408. tmr_flag_clear function

Name	Description
Function name	<code>tmr_flag_clear</code>
Function prototype	<code>void tmr_flag_clear(tmr_type *tmr_x, uint32_t tmr_flag);</code>
Function description	Clear flag
Input parameter 1	<code>tmr_x</code> : indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Input parameter 2	<code>tmr_flag</code> : flag selection Refer to tmr_flag for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_flag_clear(TMR1, TMR_OVF_FLAG);
```

5.16.46 tmr_event_sw_trigger function

The table below describes the function `tmr_event_sw_trigger`

Table 409. tmr_event_sw_trigger function

Name	Description
Function name	<code>tmr_event_sw_trigger</code>
Function prototype	<code>void tmr_event_sw_trigger(tmr_type *tmr_x, tmr_event_trigger_type tmr_event);</code>
Function description	Software triggers TMR events
Input parameter 1	<code>tmr_x</code> : indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR6, TMR14, TMR15, TMR16, TMR17
Input parameter 2	<code>tmr_event</code> : select a TMR event to be triggered by software
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);
```

5.16.47 tmr_output_enable function

The table below describes the function tmr_output_enable

Table 410. tmr_output_enable function

Name	Description
Function name	tmr_output_enable
Function prototype	void tmr_output_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR output
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR15, TMR16, TMR17
Input parameter 2	new_state: TMR output status This parameter can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_output_enable(TMR1, TRUE);
```

5.16.48 tmr_internal_clock_set function

The table below describes the function tmr_internal_clock_set.

Table 411. tmr_internal_clock_set function

Name	Description
Function name	tmr_internal_clock_set
Function prototype	void tmr_internal_clock_set(tmr_type *tmr_x);
Function description	Set TMR internal clock
Input parameter	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR15
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
tmr_internal_clock_set(TMR1);
```

5.16.49 tmr_output_channel_polarity_set function

The table below describes the function `tmr_output_channel_polarity_set`.

Table 412. tmr_output_channel_polarity_set function

Name	Description
Function name	<code>tmr_output_channel_polarity_set</code>
Function prototype	<code>void tmr_output_channel_polarity_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, tmr_polarity_active_type oc_polarity);</code>
Function description	Set TMR output channel polarity
Input parameter 1	<code>tmr_x</code> : indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR14, TMR15, TMR16, TMR17
Input parameter 2	<code>tmr_channel</code> : Timer channel
Input parameter 3	<code>oc_polarity</code> : output channel polarity
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel.

<code>TMR_SELECT_CHANNEL_1</code> :	Timer channel 1
<code>TMR_SELECT_CHANNEL_1C</code> :	Complementary channel 1
<code>TMR_SELECT_CHANNEL_2</code> :	Timer channel 2
<code>TMR_SELECT_CHANNEL_2C</code> :	Complementary channel 2
<code>TMR_SELECT_CHANNEL_3</code> :	Timer channel 3
<code>TMR_SELECT_CHANNEL_3C</code> :	Complementary channel 3
<code>TMR_SELECT_CHANNEL_4</code> :	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);
```

5.16.50 tmr_external_clock_config function

The table below describes the function tmr_external_clock_config.

Table 413. tmr_external_clock_config function

Name	Description
Function name	tmr_external_clock_config
Function prototype	void tmr_external_clock_config(tmr_type *tmr_x, tmr_external_signal_divider_type es_divide, tmr_external_signal_polarity_type es_polarity, uint16_t es_filter);
Function description	Configure TMR external clock
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3
Input parameter 2	es_divide: external signal frequency division factor
Input parameter 3	es_polarity: external signal polarity
Input parameter 4	es_filter: external signal filter value, 0x00~0x0F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

es_divide

Set TMR external signal frequency division factor.

TMR_ES_FREQUENCY_DIV_1: Divided by 1

TMR_ES_FREQUENCY_DIV_2: Divided by 2

TMR_ES_FREQUENCY_DIV_4: Divided by 4

TMR_ES_FREQUENCY_DIV_8: Divided by 8

es_polarity

Select TMR external signal polarity.

TMR_ES_POLARITY_NON_INVERTED: High or rising edge

TMR_ES_POLARITY_INVERTED: Low or falling edge

Example:

```
tmr_external_clock_config(TMR1, TMR_ES_FREQUENCY_DIV_1, TMR_ES_POLARITY_INVERTED, 0x0F);
```

5.16.51 tmr_external_clock_mode1_config function

The table below describes the function tmr_external_clock_mode1_config.

Table 414. tmr_external_clock_mode1_config function

Name	Description
Function name	tmr_external_clock_mode1_config
Function prototype	void tmr_external_clock_mode1_config(tmr_type *tmr_x, tmr_external_signal_divider_type es_divide, tmr_external_signal_polarity_type es_polarity, uint16_t es_filter);
Function description	Configure TMR external clock mode 1 (corresponding to external mode A in the reference manual)
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR15, TMR16, TMR17
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);
```

5.16.52 tmr_external_clock_mode2_config function

The table below describes the function `tmr_external_clock_mode2_config`.

Table 415. tmr_external_clock_mode2_config function

Name	Description
Function name	<code>tmr_external_clock_mode2_config</code>
Function prototype	<code>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);</code>
Function description	Configure TMR external clock mode 2 (corresponding to external mode B in the reference manual)
Input parameter 1	<code>tmr_x</code> : indicates the selected TMR peripheral, it can be TMR1, TMR3
Input parameter 2	<code>es_divide</code> : external signal frequency division factor
Input parameter 3	<code>es_polarity</code> : external signal polarity
Input parameter 4	<code>es_filter</code> : external signal filter value, 0x00~0x0F
Output parameter	NA
Return value	NA
Input parameter 2	<code>es_divide</code> : external signal frequency division factor
Input parameter 3	<code>es_polarity</code> : external signal polarity

es_divide

Set TMR external signal frequency division factor, refer to [es_divide](#) for details.

es_polarity

Set TMR external signal polarity, refer to [es_polarity](#) for details.

Example:

```
tmr_external_clock_mode2_config(TMR1, TMR_ES_FREQUENCY_DIV_1, TMR_ES_POLARITY_INVERTED, 0x0F);
```

5.16.53 tmr_encoder_mode_config function

The table below describes the function tmr_encoder_mode_config.

Table 416. tmr_encoder_mode_config function

Name	Description
Function name	tmr_encoder_mode_config
Function prototype	void tmr_encoder_mode_config(tmr_type *tmr_x, tmr_encoder_mode_type encoder_mode, tmr_input_polarity_type ic1_polarity, tmr_input_polarity_type ic2_polarity);
Function description	Configure TMR encoder mode
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3
Input parameter 2	encoder_mode: encoder mode
Input parameter 3	ic1_polarity: input channel 1 polarity
Input parameter 4	ic2_polarity: input channel 2 polarity
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

encoder_mode

Select a TMR encoder mode.

TMR_ENCODER_MODE_A: Encoder mode A

TMR_ENCODER_MODE_B: Encoder mode B

TMR_ENCODER_MODE_C: Encoder mode C

ic1_polarity

Select TMR input channel 1 polarity.

TMR_INPUT_RISING_EDGE: Rising edge

TMR_INPUT_FALLING_EDGE: Falling edge

TMR_INPUT_BOTH_EDGE: Both edges (Rising edge and Falling edge)

ic2_polarity

Select TMR input channel 2 polarity.

TMR_INPUT_RISING_EDGE: Rising edge

TMR_INPUT_FALLING_EDGE: Falling edge

TMR_INPUT_BOTH_EDGE: Both edges (Rising edge and Falling edge)

Example:

```
tmr_encoder_mode_config(TMR1, TMR_ENCODER_MODE_A, TMR_INPUT_RISING_EDGE,
TMR_INPUT_RISING_EDGE);
```


5.16.54 tmr_force_output_set function

The table below describes the function tmr_force_output_set.

Table 417. tmr_force_output_set function

Name	Description
Function name	tmr_force_output_set
Function prototype	void tmr_force_output_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, tmr_force_output_type force_output);
Function description	Set TMR forced output
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR14, TMR15, TMR16, TMR17
Input parameter 2	tmr_channel: timer channel
Input parameter 3	force_output: forced output level
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_channel

Select a TMR channel.

TMR_SELECT_CHANNEL_1: Timer channel 1

TMR_SELECT_CHANNEL_2: Timer channel 2

TMR_SELECT_CHANNEL_3: Timer channel 3

TMR_SELECT_CHANNEL_4: Timer channel 4

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);
```

5.16.55 tmr_dma_control_config function

The table below describes the function tmr_dma_control_config.

Table 418. tmr_dma_control_config function

Name	Description
Function name	tmr_dma_control_config
Function prototype	void tmr_dma_control_config(tmr_type *tmr_x, tmr_dma_transfer_length_type dma_length, tmr_dma_address_type dma_base_address);
Function description	Configure TMR DMA control
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR3, TMR15, TMR16, TMR17
Input parameter 2	dma_length: DMA transfer length
Input parameter 3	dma_base_address: DMA transfer offset address
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

dma_length

Set DAM transfer bytes, including:

TMR_DMA_TRANSFER_1BYTE: 1 byte

TMR_DMA_TRANSFER_2BYTES: 2 bytes

TMR_DMA_TRANSFER_3BYTES: 3 bytes

...

TMR_DMA_TRANSFER_17BYTES: 17 bytes

TMR_DMA_TRANSFER_18BYTES: 18 bytes

dma_base_address

Set DMA transfer offset address, starting from TMR control register 1, including:

TMR_CTRL1_ADDRESS

TMR_CTRL2_ADDRESS

TMR_STCTRL_ADDRESS

TMR_IDEN_ADDRESS

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);
```

5.16.56 tmr_brkdt_config function

The table below describes the function tmr_brkdt_config.

Table 419. tmr_brkdt_config function

Name	Description
Function name	tmr_brkdt_config
Function prototype	void tmr_brkdt_config(tmr_type *tmr_x, tmr_brkdt_config_type *brkdt_struct);
Function description	Configure TMR break mode and dead time
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR15, TMR16, TMR17
Input parameter 2	brkdt_struct: tmr_brkdt_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr_brkdt_config_type structure

The tmr_brkdt_config_type is defined in the at32l021_tmr.h:

typedef struct

```
{
    uint8_t            deadtime;
    tmr_brk_polarity_type brk_polarity;
    tmr_wp_level_type  wp_level;
    confirm_state      auto_output_enable;
    confirm_state      fcsoen_state;
    confirm_state      fcsodis_state;
    confirm_state      brk_enable;
} tmr_brkdt_config_type;
```

deadtime

Set dead time, between 0x00~0xFF

brk_polarity

Select break input polarity

TMR_BRK_INPUT_ACTIVE_LOW: Active low

TMR_BRK_INPUT_ACTIVE_HIGH: Active high

wp_level

Set write protection level.

TMR_WP_OFF: Write protection OFF

TMR_WP_LEVEL_3:

Level 3 write protection, protecting the bits below:

- TMRx_BRK: DTC, BRKEN, BRKV and AOEN
- TMRx_CTRL2: CxIOS and CxCIOS

TMR_WP_LEVEL_2:

Level 2 write protection, protecting the bits below in addition to level-3 protected bits:

- TMRx_CCTRL: CxP and CxCP
- TMRx_BRK: FCSODIS and FCSOEN

TMR_WP_LEVEL_1:

Level 1 write protection, protecting the bits below in addition to level-2 protected bits:

- TMRx_CMx: CxOCTRL and CxOBEN

auto_output_enable

Enable auto output, Enable (TRUE) or disable (FALSE)

fcsoen_state

Indicates the frozen status when main output is ON. It is used to configure the status of complementary output channels when timer is OFF and output is enabled (OEN=1).

FALSE: Disable CxOUT/CxCOUT output

TRUE: Enable CxOUT/CxCOUT output, inactive level

fcsodis_state

Indicates the frozen status when main output is OFF. It is used to configure the status of complementary output channels when timer is OFF and output is disabled (OEN=0).

FALSE: Disable CxOUT/CxCOUT output

TRUE: Enable CxOUT/CxCOUT output, idle level

brk_enable

Enable break feature, Enable (TRUE) or disable (FALSE).

Example

```
tmr_brkdt_config_type tmr_brkdt_config_struct;
tmr_brkdt_config_struct.brk_enable = TRUE;
tmr_brkdt_config_struct.auto_output_enable = TRUE;
tmr_brkdt_config_struct.deadtime = 0;
tmr_brkdt_config_struct.fcsoen_state = TRUE;
tmr_brkdt_config_struct.fcsodis_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);
```

5.16.57 tmr_brk_filter_value_set function

The table below describes the function tmr_brk_filter_value_set.

Table 420. tmr_brk_filter_value_set function

Name	Description
Function name	tmr_brk_filter_value_set
Function prototype	void tmr_brk_filter_value_set(tmr_type *tmr_x, uint8_t filter_value)
Function description	Set TMR break input filter value
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR15, TMR16, TMR17
Input parameter 2	filter_value: filter value (from 0x0 to 0xf)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example

```
tmr_brk_filter_value_set (TMR1, 0x4);
```

5.16.58 tmr_iremap_config function

The table below describes the function tmr_iremap_config.

Table 421. tmr_iremap_config function

Name	Description
Function name	tmr_iremap_config
Function prototype	void tmr_iremap_config(tmr_type *tmr_x, tmr_input_remap_type input_remap);
Function description	Set TMR internal remapping
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR14
Input parameter 2	input_remap: TMR input channel remap to be configured
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

input_remap

Set TMR2 internal trigger 1 remapping and TMR14 channel 1 input remapping

TMR14_GPIO: TMR14 channel 1 is connected to GPIO

TMR14_ERTCCLK: TMR14 channel 1 is connected to ERTC

TMR14_HEXT_DIV32: TMR14 channel 1 is connected to HEXT/32

TMR14_CLKOUT: TMR14 channel 1 is connected to CLKOUT

Example

```
tmr_iremap_config(TMR14, TMR14_LICK);
```

5.17 Watchdog timer (WDT)

The WDT register structure `wdt_type` is defined in the “at32l021_wdt.h”:

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

} wdt_type;
```

The table below gives a list of the WDT registers

Table 422. Summary of WDT registers

Register	Description
cmd	Command register
div	Divider register
rl	Reload register
sts	Status register
win	Window register

The table below gives a list of WDT library functions.

Table 423. Summary of WDT library functions

Function name	Description
<code>wdt_enable</code>	Enable watchdog
<code>wdt_counter_reload</code>	Reload counter
<code>wdt_reload_value_set</code>	Set reload value
<code>wdt_divider_set</code>	Set division value
<code>wdt_register_write_enable</code>	Unlock WDT_DIV, WDT_RLD and WDT_WIN register write protection
<code>wdt_flag_get</code>	Get flag
<code>wdt_window_counter_set</code>	Set window register

5.17.1 wdt_enable function

The table below describes the function wdt_enable.

Table 424. wdt_enable function

Name	Description
Function name	wdt_enable
Function prototype	void wdt_enable(void);
Function description	Enable watchdog
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wdt_enable();
```

5.17.2 wdt_counter_reload function

The table below describes the function wdt_counter_reload.

Table 425. wdt_counter_reload function

Name	Description
Function name	wdt_counter_reload
Function prototype	void wdt_counter_reload(void);
Function description	Reload counter
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wdt_counter_reload();
```

5.17.3 wdt_reload_value_set function

The table below describes the function wdt_reload_value_set.

Table 426. 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	reload_value: reload value, 0x000~0xFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wdt_reload_value_set(0xFFFF);
```

5.17.4 wdt_divider_set function

The table below describes the function wdt_divider_set.

Table 427. 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	Division: watchdog division value Refer to the "division" description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

division

Select watchdog division value.

WDT_CLK_DIV_4: Divided by 4

WDT_CLK_DIV_8: Divided by 8

WDT_CLK_DIV_16: Divided by 16

WDT_CLK_DIV_32: Divided by 32

WDT_CLK_DIV_64: Divided by 64

WDT_CLK_DIV_128: Divided by 128

WDT_CLK_DIV_256: Divided by 256

Example:

```
wdt_divider_set(WDT_CLK_DIV_4);
```


5.17.5 wdt_register_write_enable function

The table below describes the function wdt_register_write_enable.

Table 428. 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 DT_WIN write protection
Input parameter	new_state: unlock register write protection This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wdt_register_write_enable(TRUE);
```

5.17.6 wdt_flag_get function

The table below describes the function wdt_flag_get.

Table 429. 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	Flag: flag selection Refer to the “flag” description below for details.
Output parameter	NA
Return value	flag_status: flag status This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

flag

This is used for flag selection, including:

WDT_DIVF_UPDATE_FLAG: Division value update complete
WDT_RLDF_UPDATE_FLAG: Reload value update complete
WDT_WINF_UPDATE_FLAG: Window value update complete

Example:

```
wdt_flag_get(WDT_DIVF_UPDATE_FLAG);
```

5.17.7 wdt_window_counter_set function

The table below describes the function wdt_window_counter_set.

Table 430. 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 counter
Input parameter	window_cnt: window value, from 0x000 to 0xFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wdt_window_counter_set(0x7FF);
```

5.18 Universal synchronous/asynchronous receiver/transmitter (USART)

The USART register structure `usart_type` is defined in the “at32l021_usart.h”:

```
/**
 * @brief type define usart register all
 */
typedef struct
{
    ...
} usart_type;
```

The table below gives a list of the USART registers

Table 431. Summary of USART registers

Register	Description
sts	Status register
dt	Data register
baudr	Baud rate register
ctrl1	Control register 1
ctrl2	Control register 2
ctrl3	Control register 3
gdiv	Guard time and divider Control register 1
rtov	Receiver timeout detection value register
ifc	Interrupt flag bit clear register

The table below gives a list of USART library functions.

Table 432. Summary of USART library functions

Function name	Description
<code>usart_reset</code>	Reset USART peripheral registers
<code>usart_init</code>	Set baud rate, data bits and stop bits.
<code>usart_parity_selection_config</code>	Parity selection
<code>usart_enable</code>	Enable USART peripherals
<code>usart_transmitter_enable</code>	Enable USART transmitter
<code>usart_receiver_enable</code>	Enable USART receiver
<code>usart_clock_config</code>	Set clock polarity and phases for synchronization
<code>usart_clock_enable</code>	Set clock output for synchronization
<code>usart_interrupt_enable</code>	Enable interrupts
<code>usart_dma_transmitter_enable</code>	Enable DMA transmitter
<code>usart_dma_receiver_enable</code>	Enable DMA receiver
<code>usart_wakeup_id_set</code>	Set wakeup ID
<code>usart_wakeup_mode_set</code>	Set wakeup mode
<code>usart_receiver_mute_enable</code>	Enable receiver mute mode
<code>usart_break_bit_num_set</code>	Set break frame length
<code>usart_lin_mode_enable</code>	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_flag_clear	Clear flag
usart_rs485_delay_time_config	Set latency for starting or ending consecutive data transmission in RS 485
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
usart_low_power_wakeup_set	Set low-power wakeup
usart_deep_sleep_mode_enable	Deepsleep mode enable
usart_msb_transmit_first_enable	Enable MSB-first transmission
usart_dt_polarity_reverse	Enable data polarity reverse mode
usart_transmit_pin_polarity_reverse	Enable data transmit pin polarity reverse mode
usart_receive_pin_polarity_reverse	Enable data receive pin polarity reverse mode
usart_receiver_timeout_detection_enable	Enable data receive timeout detection
usart_receiver_timeout_value_set	Set data receive timeout value

5.18.1 usart_reset function

The table below describes the function usart_reset.

Table 433. usart_reset function

Name	Description
Function name	usart_reset
Function prototype	void usart_reset(usart_type* usart_x);
Function description	Reset USART peripheral registers
Input parameter 1	usart_x: indicates the selected peripherals, it can be USART1, USART2, 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);
```

5.18.2 usart_init function

The table below describes the function usart_init.

Table 434. usart_init function

Name	Description
Function name	usart_init
Function prototype	void usart_init(usart_type* usart_x, uint32_t baud_rate, usart_data_bit_num_type data_bit, usart_stop_bit_num_type stop_bit);
Function description	Set baud rate, data bits and stop bits.
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, 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);
```

5.18.3 usart_parity_selection_config function

The table below describes the function usart_parity_selection_config.

Table 435. 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 method selection
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, or USART3...
Input parameter 2	Parity: parity mode for serial interface communication
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

parity

Select parity mode for serial interface communication.

USART_PARITY_NONE: No parity

USART_PARITY_EVEN: Even

USART_PARITY_ODD: Odd

Example:

```
/* config usart even parity */  
usart_parity_selection_config(USART1, USART_PARITY_EVEN);
```

5.18.4 usart_enable function

The table below describes the function usart_enable.

Table 436. usart_enable function

Name	Description
Function name	usart_enable
Function prototype	void usart_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable USART
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, 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);
```

5.18.5 usart_transmitter_enable function

The table below describes the function usart_transmitter_enable.

Table 437. usart_transmitter_enable function

Name	Description
Function name	usart_transmitter_enable
Function prototype	void usart_transmitter_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable USART transmitter
Input parameter 1	usart_x: indicates the selected peripheral it can be USART1, USART2, 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 transmitter */  
usart_transmitter_enable(USART1, TRUE);
```

5.18.6 usart_receiver_enable function

The table below describes the function usart_receiver_enable.

Table 438. usart_receiver_enable function

Name	Description
Function name	usart_receiver_enable
Function prototype	void usart_receiver_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable USART receiver
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, 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);
```

5.18.7 usart_clock_config function

The table below describes the function usart_clock_config.

Table 439. 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_lbc_type clk_lb);
Function description	Configure clock polarity and phase for synchronization feature
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, 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);
```

5.18.8 usart_clock_enable function

The table below describes the function usart_clock_enable.

Table 440. usart_clock_enable function

Name	Description
Function name	usart_clock_enable
Function prototype	void usart_clock_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable clock output
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, 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);
```

5.18.9 usart_interrupt_enable function

The table below describes the function usart_interrupt_enable.

Table 441. usart_interrupt_enable function

Name	Description
Function name	usart_interrupt_enable
Function prototype	void usart_interrupt_enable(usart_type* usart_x, uint32_t usart_int, confirm_state new_state);
Function description	Enable interrupts
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, or USART3...
Input parameter 2	usart_int: interrupt type
Input parameter 3	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

usart_int

Defines a peripheral interrupt.

USART_IDLE_INT:	Bus idle
USART_RDBF_INT:	Receive data buffer full
USART_TDC_INT:	Transmit data complete
USART_TDBE_INT:	Transmit data buffer empty
USART_PERR_INT:	Parity error
USART_BF_INT:	Break frame receive
USART_ERR_INT:	Error interrupt
USART_CTSCF_INT:	CTS (Clear To Send) change
USART_LPWUF_INT:	Low-power wakeup interrupt
USART_RTOD_INT:	Receive timeout detection interrupt
USART_CMD_INT:	Character match detection interrupt

Example:

```
/* enable usart1 transmit complete interrupt */
usart_interrupt_enable (USART1, USART_TDC_INT, TRUE);
```

5.18.10 usart_dma_transmitter_enable function

The table below describes the function usart_dma_transmitter_enable.

Table 442. usart_dma_transmitter_enable function

Name	Description
Function name	usart_dma_transmitter_enable
Function prototype	void usart_dma_transmitter_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable DMA transmitter
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, 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);
```

5.18.11 usart_dma_receiver_enable function

The table below describes the function usart_dma_receiver_enable.

Table 443. usart_dma_receiver_enable function

Name	Description
Function name	usart_dma_receiver_enable
Function prototype	void usart_dma_receiver_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable DMA receiver
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, 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);
```

5.18.12 usart_wakeup_id_set function

The table below describes the function usart_wakeup_id_set.

Table 444. usart_wakeup_id_set function

Name	Description
Function name	usart_wakeup_id_set
Function prototype	void usart_wakeup_id_set(usart_type* usart_x, uint8_t usart_id);
Function description	Set wakeup ID
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, 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);
```

5.18.13 usart_wakeup_mode_set function

The table below describes the function usart_wakeup_mode_set.

Table 445. usart_wakeup_mode_set function

Name	Description
Function name	usart_wakeup_mode_set
Function prototype	void usart_wakeup_mode_set(usart_type* usart_x, usart_wakeup_mode_type wakeup_mode);
Function description	Set wakeup mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, 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);
```

5.18.14 usart_receiver_mute_enable function

The table below describes the function usart_receiver_mute_enable.

Table 446. usart_receiver_mute_enable function

Name	Description
Function name	usart_receiver_mute_enable
Function prototype	void usart_receiver_mute_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable USART receiver mute mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, 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);
```

5.18.15 usart_break_bit_num_set function

The table below describes the function usart_break_bit_num_set.

Table 447. usart_break_bit_num_set function

Name	Description
Function name	usart_break_bit_num_set
Function prototype	void usart_break_bit_num_set(usart_type* usart_x, usart_break_bit_num_type break_bit);
Function description	Set USART break frame length
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, 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);
```

5.18.16 usart_lin_mode_enable function

The table below describes the function usart_lin_mode_enable.

Table 448. usart_lin_mode_enable function

Name	Description
Function name	usart_lin_mode_enable
Function prototype	void usart_lin_mode_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable LIN mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, 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);
```

5.18.17 usart_data_transmit function

The table below describes the function usart_data_transmit.

Table 449. usart_data_transmit function

Name	Description
Function name	usart_data_transmit
Function prototype	void usart_data_transmit(usart_type* usart_x, uint16_t data);
Function description	Transmit data
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, or USART3
Input parameter 2	Data: data to be sent
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* transmit data */
uint16_t data = 0x88;
usart_data_transmit (USART1, data);
```

5.18.18 usart_data_receive function

The table below describes the function usart_data_receive.

Table 450. 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: indicates the 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);
```

5.18.19 usart_break_send function

The table below describes the function usart_break_send.

Table 451. usart_break_send function

Name	Description
Function name	usart_break_send
Function prototype	void usart_break_send(usart_type* usart_x);
Function description	Sends break frame
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, 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);
```

5.18.20 usart_smartcard_guard_time_set function

The table below describes the function usart_smartcard_guard_time_set.

Table 452. usart_smartcard_guard_time_set function

Name	Description
Function name	usart_smartcard_guard_time_set
Function prototype	void usart_smartcard_guard_time_set(usart_type* usart_x, uint8_t guard_time_val);
Function description	Set smartcard guard time
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, 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);
```


5.18.21 usart_irda_smartcard_division_set function

The table below describes the function usart_irda_smartcard_division_set.

Table 453. usart_irda_smartcard_division_set function

Name	Description
Function name	usart_irda_smartcard_division_set
Function prototype	void usart_irda_smartcard_division_set(usart_type* usart_x, uint8_t div_val);
Function description	Infrared and smartcard frequency division settings
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 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);
```

5.18.22 usart_smartcard_mode_enable function

The table below describes the function usart_smartcard_mode_enable.

Table 454. usart_smartcard_mode_enable function

Name	Description
Function name	usart_smartcard_mode_enable
Function prototype	void usart_smartcard_mode_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable smartcode mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 or USART3
Input parameter 2	new_state: Enable (TRUE), Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable the smartcard mode */
usart_smartcard_mode_enable(USART1, TRUE);
```

5.18.23 usart_smartcard_nack_set function

The table below describes the function usart_smartcard_nack_set.

Table 455. usart_smartcard_nack_set function

Name	Description
Function name	usart_smartcard_nack_set
Function prototype	void usart_smartcard_nack_set(usart_type* usart_x, confirm_state new_state);
Function description	Enable smartcard NACK
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 or USART3
Input parameter 2	new_state: Enable (TRUE), Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable the nack transmission */
usart_smartcard_nack_set(USART1, TRUE);
```

5.18.24 usart_single_line_halfduplex_select function

The table below describes the function usart_single_line_halfduplex_select.

Table 456. usart_single_line_halfduplex_select function

Name	Description
Function name	usart_single_line_halfduplex_select
Function prototype	void usart_single_line_halfduplex_select(usart_type* usart_x, confirm_state new_state);
Function description	Enable single-wire half-duplex mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 or USART3
Input parameter 2	new_state: Enable (TRUE), Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable halfduplex */
usart_single_line_halfduplex_select(USART1, TRUE);
```

5.18.25 usart_irda_mode_enable function

The table below describes the function usart_irda_mode_enable.

Table 457. usart_irda_mode_enable function

Name	Description
Function name	usart_irda_mode_enable
Function prototype	void usart_irda_mode_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable infrared mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 or USART3
Input parameter 2	new_state: Enable (TRUE), Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable irda mode */
usart_irda_mode_enable(USART1, TRUE);
```

5.18.26 usart_irda_low_power_enable function

The table below describes the function usart_irda_low_power_enable.

Table 458. usart_irda_low_power_enable function

Name	Description
Function name	usart_irda_low_power_enable
Function prototype	void usart_irda_low_power_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable infrared low-power mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 or USART3
Input parameter 2	new_state: Enable (TRUE), Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable irda lowpower mode */
usart_irda_low_power_enable (USART1, TRUE);
```

5.18.27 usart_hardware_flow_control_set function

The table below describes the function usart_hardware_flow_control_set.

Table 459. usart_hardware_flow_control_set function

Name	Description
Function name	usart_hardware_flow_control_set
Function prototype	void usart_hardware_flow_control_set(usart_type* usart_x, usart_hardware_flow_control_type flow_state);
Function description	Set peripheral hardware flow control
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 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);
```

5.18.28 usart_flag_get function

The table below describes the function usart_flag_get.

Table 460. usart_flag_get function

Name	Description
Function name	usart_flag_get
Function prototype	flag_status usart_flag_get(usart_type* usart_x, uint32_t flag);
Function description	Get flag status
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 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_RTODF_FLAG: Receiver timeout detection flag
 USART_OCCUPY_FLAG: Receiver occupied flag
 USART_CMDF_FLAG: Character match detection flag
 USART_LPWUF_FLAG: Low-power wakeup flag
 USART_TXON_FLAG: Transmitter complete flag

USART_RXON_FLAG:	Receiver complete flag
USART_CTSCF_FLAG:	CTS (Clear To Send) change flag
USART_BFF_FLAG:	Break frame receive flag
USART_TDBE_FLAG:	Transmit buffer empty flag
USART_TDC_FLAG:	Transmit complete flag
USART_RDBF_FLAG:	Receive data buffer full flag
USART_IDLEF_FLAG:	Idle frame flag
USART_ROERR_FLAG:	Receive overflow flag
USART_NERR_FLAG:	Noise error flag
USART_FERR_FLAG:	Frame error flag
USART_PERR_FLAG:	Parity error flag

Example:

```
/* wait data transmit complete flag */
while(usart_flag_get(USART1, USART_TDC_FLAG) == RESET);
```

5.18.29 usart_interrupt_flag_get function

The table below describes the function usart_interrupt_flag_get.

Table 461. 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 interrupt flag status
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2...
Input parameter 2	Flag: clear the selected flag
Output parameter	NA
Return value	flag_status: Return SET or RESE
Required preconditions	NA
Called functions	NA

Flag

USART_RTODF_FLAG:	Receiver timeout detection flag
USART_CMDF_FLAG:	Character match detection flag
USART_LPWUF_FLAG:	Low-power wakeup flag
USART_CTSCF_FLAG:	CTS (Clear To Send) change flag
USART_BFF_FLAG:	Break frame receive flag
USART_TDBE_FLAG:	Transmit buffer empty flag
USART_TDC_FLAG:	Transmit complete flag
USART_RDBF_FLAG:	Receive data buffer full flag
USART_IDLEF_FLAG:	Idle frame flag
USART_ROERR_FLAG:	Receive overflow flag
USART_NERR_FLAG:	Noise error flag
USART_FERR_FLAG:	Frame error flag
USART_PERR_FLAG:	Parity error flag

Example:

```
/* check received data flag */
if(usart_interrupt_flag_get(USART1, USART_RDBF_FLAG) != RESET)
```

```
{
}
```

5.18.30 usart_flag_clear function

The table below describes the function usart_flag_clear.

Table 462. usart_flag_clear function

Name	Description
Function name	usart_flag_clear
Function prototype	void usart_flag_clear(usart_type* usart_x, uint32_t flag);
Function description	Clear flag
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 or USART3
Input parameter 2	Flag: clear the selected flag
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

flag

USART_RTODF_FLAG:	Receiver timeout detection flag
USART_OCCUPY_FLAG:	Receiver occupied flag
USART_CMDF_FLAG:	Character match detection flag
USART_LPWUF_FLAG:	Low-power wakeup flag
USART_TXON_FLAG:	Transmitter complete flag
USART_RXON_FLAG:	Receiver complete flag
USART_CTSCF_FLAG:	CTS (Clear To Send) change flag
USART_BFF_FLAG:	Break frame receive 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:

```
/* clear data receive complete flag */
usart_flag_clear (USART1, USART_RDBF_FLAG );
```

5.18.31 usart_rs485_delay_time_config function

The table below describes the function usart_rs485_delay_time_config.

Table 463. usart_flag_clear function

Name	Description
Function name	usart_rs485_delay_time_config
Function prototype	void usart_rs485_delay_time_config(usart_type* usart_x, uint8_t start_delay_time, uint8_t complete_delay_time);
Function description	Delay time for setting and clearing data valid signal for continuous data transmission in RS485 mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	start_delay_time: 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);
```

5.18.32 usart_transmit_receive_pin_swap function

The table below describes the function usart_transmit_receive_pin_swap.

Table 464. usart_transmit_receive_pin_swap function

Name	Description
Function name	usart_transmit_receive_pin_swap
Function prototype	void usart_transmit_receive_pin_swap(usart_type* usart_x, confirm_state new_state);
Function description	Swap transmit/receive pins
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	new_state: enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable tx/rx swap */
usart_transmit_receive_pin_swap (USART1, TRUE);
```

5.18.33 usart_id_bit_num_set function

The table below describes the function usart_id_bit_num_set.

Table 465. usart_transmit_receive_pin_swap function

Name	Description
Function name	usart_id_bit_num_set
Function prototype	void usart_id_bit_num_set(usart_type* usart_x, usart_identification_bit_num_type id_bit_num);
Function description	Set ID bit number
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	id_bit_num: ID bit count
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

id_bit_num

USART_ID_FIXED_4_BIT: 4-bit ID

USART_ID_RELATED_DATA_BIT: Current data bit -1

Example:

```
/* config ID bit width */
usart_id_bit_num_set(USART1, USART_ID_FIXED_4_BIT);
```

5.18.34 usart_de_polarity_set function

The table below describes the function usart_de_polarity_set.

Table 466. 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	Set DE signal polarity
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	new_state: Enable (TRUE), disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

de_polarity

USART_DE_POLARITY_HIGH: DE signal high level active

USART_DE_POLARITY_LOW: DE signal low level active

Example:

```
/* config DE polarity */
usart_de_polarity_set(USART1, USART_DE_POLARITY_HIGH);
```


5.18.35 usart_rs485_mode_enable function

The table below describes the function usart_rs485_mode_enable.

Table 467. usart_rs485_mode_enable function

Name	Description
Function name	usart_rs485_mode_enable
Function prototype	void usart_rs485_mode_enable(usart_type* usart_x, confirm_state new_state);
Function description	RS485 mode enable
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	new_state: enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable rs485 mode */  
usart_rs485_mode_enable (USART1, TRUE);
```

5.18.36 usart_low_power_wakeup_set function

The table below describes the function usart_low_power_wakeup_set.

Table 468. usart_low_power_wakeup_set function

Name	Description
Function name	usart_low_power_wakeup_set
Function prototype	void usart_low_power_wakeup_set(usart_type* usart_x, usart_wakeup_method_type wakeup_method);
Function description	Set low-power wakeup mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	new_state: enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

wakeup_method

USART_WAKEUP_METHOD_ID: Receive match ID wakeup mode

USART_WAKEUP_METHOD_START: Start bit detect wakeup mode

USART_WAKEUP_METHOD_RDBF: Receive data buff full wakeup mode

Example:

```
/* set wakeup method */
usart_low_power_wakeup_set(USART1, USART_WAKEUP_METHOD_RDBF);
```

5.18.37 usart_deep_sleep_mode_enable function

The table below describes the function usart_deep_sleep_mode_enable.

Table 469. usart_deep_sleep_mode_enable function

Name	Description
Function name	usart_deep_sleep_mode_enable
Function prototype	void usart_deep_sleep_mode_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable peripheral deepsleep mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	new_state: enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable deep sleep mode */
usart_deep_sleep_mode_enable(USART1, TRUE);
```

5.18.38 usart_msb_transmit_first_enable function

The table below describes the function usart_msb_transmit_first_enable.

Table 470. usart_msb_transmit_first_enable function

Name	Description
Function name	usart_msb_transmit_first_enable
Function prototype	void usart_msb_transmit_first_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable MSB (most significant bit) first transmit
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	new_state: enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable most significant bit transmit first */
usart_msb_transmit_first_enable(USART1, TRUE);
```

5.18.39 usart_dt_polarity_reverse function

The table below describes the function usart_dt_polarity_reverse

Table 471. usart_dt_polarity_reverse function

Name	Description
Function name	usart_dt_polarity_reverse
Function prototype	void usart_dt_polarity_reverse(usart_type* usart_x, confirm_state new_state);
Function description	Reverse data polarity
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	new_state: enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* reverse data polarity */
usart_dt_polarity_reverse (USART1, ENABLE);
```

5.18.40 usart_transmit_pin_polarity_reverse function

The table below describes the function usart_dt_polarity_reverse

Table 472. usart_dt_polarity_reverse function

Name	Description
Function name	usart_transmit_pin_polarity_reverse
Function prototype	void usart_transmit_pin_polarity_reverse(usart_type* usart_x, confirm_state new_state);
Function description	Reverse data transmit pin polarity
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	new_state: enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* reverse transmit pin polarity */
usart_transmit_pin_polarity_reverse (USART1, ENABLE);
```

5.18.41 usart_receive_pin_polarity_reverse function

The table below describes the function usart_receive_pin_polarity_reverse

Table 473. usart_receive_pin_polarity_reverse function

Name	Description
Function name	usart_receive_pin_polarity_reverse
Function prototype	usart_receive_pin_polarity_reverse (usart_type* usart_x, confirm_state new_state);
Function description	Reverse data receive pin polarity
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	new_state: enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* reverse receive pin polarity */
usart_receive_pin_polarity_reverse (USART1, ENABLE);
```

5.18.42 usart_receiver_timeout_detection_enable function

The table below describes the function usart_receiver_timeout_detection_enable

Table 474. usart_receiver_timeout_detection_enable function

Name	Description
Function name	usart_receiver_timeout_detection_enable
Function prototype	usart_receiver_timeout_detection_enable (usart_type* usart_x, confirm_state new_state);
Function description	Enable data receive timeout detection
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	new_state: enabled (TRUE) or disabled (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* enable receive timeout detection */
usart_receiver_timeout_detection_enable (USART1, ENABLE);
```

5.18.43 usart_receiver_timeout_value_set function

The table below describes the function usart_receiver_timeout_value_set

Table 475. usart_receiver_timeout_value_set function

Name	Description
Function name	usart_receiver_timeout_value_set
Function prototype	void usart_receiver_timeout_value_set(usart_type* usart_x, uint32_t time);
Function description	Set data receive timeout value
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	Time: Set data receive timeout threshold
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* configure receive timeout value */
usart_receiver_timeout_value_set (USART1, 32);
```

5.19 Window watchdog timer (WWDT)

The WWDT register structure `wwdt_type` is defined in the “`at32l021_wwdt.h`”:

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

} wwdt_type;
```

The table below gives a list of the WWDT registers

Table 476. 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 477. Summary of WWDT library functions

Function name	Description
<code>wwdt_reset</code>	Reset window watchdog registers
<code>wwdt_divider_set</code>	Set divider
<code>wwdt_flag_clear</code>	Clear reload counter interrupt flag
<code>wwdt_enable</code>	Enable WWDT
<code>wwdt_interrupt_enable</code>	Enable reload counter interrupt
<code>wwdt_flag_get</code>	Get flag
<code>wwdt_counter_set</code>	Set counter value
<code>wwdt_window_counter_set</code>	Set window value

5.19.1 wwdt_reset function

The table below describes the function wwdt_reset.

Table 478. wwdt_reset function

Name	Description
Function name	wwdt_reset
Function prototype	void wwdt_reset(void);
Function description	Reset window watchdog registers to their initial values.
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	void crm_periph_reset(crm_periph_reset_type value, confirm_state new_state);

Example:

```
wwdt_reset();
```

5.19.2 wwdt_divider_set function

The table below describes the function wwdt_divider_set.

Table 479. wwdt_divider_set function

Name	Description
Function name	wwdt_divider_set
Function prototype	void wwdt_divider_set(wwdt_division_type division);
Function description	Set divider
Input parameter	Division: WWDT division value Refer to the “division” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

division

Select WWDT division value.

WWDT_PCLK1_DIV_4096: Divided by 4096

WWDT_PCLK1_DIV_8192: Divided by 8192

WWDT_PCLK1_DIV_16384: Divided by 16384

WWDT_PCLK1_DIV_32768: Divided by 32768

Example:

```
wwdt_divider_set(WWDT_PCLK1_DIV_4096);
```

5.19.3 wwdt_enable function

The table below describes the function wwdt_enable.

Table 480. wwdt_enable function

Name	Description
Function name	wwdt_enable
Function prototype	void wwdt_enable(uint8_t wwdt_cnt);
Function description	Enable WWDT
Input parameter	wwdt_cnt: WWDT counter initial value, 0x40~0x7F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_enable(0x7F);
```

5.19.4 wwdt_interrupt_enable function

The table below 3 describes the function wwdt_interrupt_enable.

Table 481. wwdt_interrupt_enable function

Name	Description
Function name	wwdt_interrupt_enable
Function prototype	void wwdt_interrupt_enable(void);
Function description	Enable reload counter interrupt
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_interrupt_enable();
```

5.19.5 wwdt_counter_set function

The table below describes the function wwdt_counter_set.

Table 482. wwdt_counter_set function

Name	Description
Function name	wwdt_counter_set
Function prototype	void wwdt_counter_set(uint8_t wwdt_cnt);
Function description	Set counter value
Input parameter	wwdt_cnt: WWDT counter value, 0x40~0x7F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_counter_set(0x7F);
```

5.19.6 wwdt_window_counter_set function

The table below describes the function wwdt_window_counter_set.

Table 483. wwdt_window_counter_set function

Name	Description
Function name	wwdt_window_counter_set
Function prototype	void wwdt_window_counter_set(uint8_t window_cnt);
Function description	Set window counter value
Input parameter	wwdt_cnt: WWDT window value, 0x40~0x7F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_window_counter_set(0x6F);
```

5.19.7 wwdt_flag_get function

The table below describes the function wwdt_flag_get.

Table 484. wwdt_flag_get function

Name	Description
Function name	wwdt_flag_get
Function prototype	flag_status wwdt_flag_get(void);
Function description	Get reload counter interrupt flag
Input parameter	NA
Output parameter	NA
Return value	flag_status: flag status. Return SET or RESET
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_flag_get();
```

5.19.8 wwdt_interrupt_flag_get function

The table below describes the function `wwdt_interrupt_flag_get`.

Table 485. wwdt_interrupt_flag_get function

Name	Description
Function name	<code>wwdt_interrupt_flag_get</code>
Function prototype	<code>flag_status wwdt_interrupt_flag_get(void);</code>
Function description	Get reload counter interrupt flag status, and check the corresponding interrupt enable bit
Input parameter	NA
Output parameter	NA
Return value	<code>flag_status</code> : Return SET or RESET
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_interrupt_flag_get();
```

5.19.9 wwdt_flag_clear function

The table below describes the function `wwdt_flag_clear`.

Table 486. wwdt_flag_clear function

Name	Description
Function name	<code>wwdt_flag_clear</code>
Function prototype	<code>void wwdt_flag_clear(void);</code>
Function description	Clear reload counter interrupt flag
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
wwdt_flag_clear();
```

5.20 Power controller (PWC)

The PWC register structure pwc_type is defined in the “at32l021_pwc.h”:

```
/**
 * @brief type define pwc register all
 */
typedef struct
{
    .....
} pwc_type;
```

The table below gives a list of the PWC registers

Table 487. Summary of PWC registers

Register	Description
ctrl	Power control register
ctrlsts	Power control/status register

The table below gives a list of PWC library functions.

Table 488. 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

5.20.1 pwc_reset function

The table below describes the function pwc_reset.

Table 489. 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();
```

5.20.2 pwc_battery_powered_domain_access function

The table below describes the function pwc_battery_powered_domain_access.

Table 490. 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.

5.20.3 pwc_pvm_level_select function

The table below describes the function pwc_pvm_level_select.

Table 491. pwc_pvm_level_select function

Name	Description
Function name	pwc_pvm_level_select
Function prototype	void pwc_pvm_level_select(pwc_pvm_voltage_type pvm_voltage)
Function description	Select PVM threshold
Input parameter	pvm_voltage: indicates the selected PVM threshold This parameter can be any enumerated value in the pwc_pvm_voltage_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pvm_voltage

pvm_voltage is used to select a PVM threshold from the optional parameters below:

PWC_PVM_VOLTAGE_2V15:	PVM threshold is 2.15V
PWC_PVM_VOLTAGE_2V30:	PVM threshold is 2.30V
PWC_PVM_VOLTAGE_2V45:	PVM threshold is 2.45V
PWC_PVM_VOLTAGE_2V60:	PVM threshold is 2.60V
PWC_PVM_VOLTAGE_2V75:	PVM threshold is 2.75V
PWC_PVM_VOLTAGE_2V90:	PVM threshold is 2.90V
PWC_PVM_VOLTAGE_3V00:	PVM threshold is 3.00V

Example:

```
/* set the threshold voltage to 2.90v */
pwc_pvm_level_select(PWC_PVM_VOLTAGE_2V90);
```

5.20.4 pwc_power_voltage_monitor_enable function

The table below describes the function pwc_power_voltage_monitor_enable.

Table 492. 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);
```

5.20.5 pwc_wakeup_pin_enable function

The table below describes the function pwc_wakeup_pin_enable.

Table 493. 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

pin_num is used to select Standby-mode wakeup pin, including:

PWC_WAKEUP_PIN_1: Standby wakeup pin 1 (corresponding GPIO is PA0)

PWC_WAKEUP_PIN_2: Standby wakeup pin 1 (corresponding GPIO is PC13)

PWC_WAKEUP_PIN_4: Standby wakeup pin 4 (corresponding GPIO is PA2)

PWC_WAKEUP_PIN_6: Standby wakeup pin 6 (corresponding GPIO is PB5)

PWC_WAKEUP_PIN_7: Standby wakeup pin 7 (corresponding GPIO is PB15)

Example:

```
/* enable wakeup pin - pa0 */
pwc_wakeup_pin_enable(PWC_WAKEUP_PIN_1, TRUE);
```


5.20.6 pwc_flag_clear function

The table below describes the function pwc_flag_clear.

Table 494. 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 the “pwc_flag” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pwc_flag

pwc_flag is used to select a flag from the optional parameters below:

PWC_WAKEUP_FLAG: Standby wakeup event

PWC_STANDBY_FLAG: Standby mode entry

PWC_PVM_OUTPUT_FLAG: PVM output (this parameter cannot be cleared by software)

Example:

```
/* wakeup event flag clear */
pwc_flag_clear(PWC_WAKEUP_FLAG);
```

5.20.7 pwc_flag_get function

The table below describes the function pwc_flag_get.

Table 495. 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 for details.
Output parameter	NA
Return value	flag_status: indicates flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

Example:

```
/* check if wakeup event flag is set */
if(pwc_flag_get(PWC_WAKEUP_FLAG) != RESET)
```

5.20.8 pwc_sleep_mode_enter function

The table below describes the function pwc_sleep_mode_enter.

Table 496. pwc_sleep_mode_enter function

Name	Description
Function name	pwc_sleep_mode_enter
Function prototype	void pwc_sleep_mode_enter(pwc_sleep_enter_type pwc_sleep_enter)
Function description	Enter Sleep mode
Input parameter	pwc_sleep_enter: select a command to enter Sleep mode This parameter can be any enumerated value in the pwc_sleep_enter_type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pwc_sleep_enter

pwc_sleep_enter is used to select a command to enter Sleep mode 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);
```

5.20.9 pwc_deep_sleep_mode_enter function

The table below describes the function pwc_deep_sleep_mode_enter.

Table 497. pwc_deep_sleep_mode_enter function

Name	Description
Function name	pwc_deep_sleep_mode_enter
Function prototype	void pwc_deep_sleep_mode_enter(pwc_deep_sleep_enter_type pwc_deep_sleep_enter)
Function description	Enter Deepsleep mode
Input parameter	pwc_deep_sleep_enter: select a command to enter Deepsleep mode This parameter can be any enumerated value in the pwc_deep_sleep_enter_type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

pwc_deep_sleep_enter

pwc_deep_sleep_enter is used to select a command to enter Deepsleep mode, including:

PWC_DEEP_SLEEP_ENTER_WFI: Enter Deepsleep mode by WFI

PWC_DEEP_SLEEP_ENTER_WFE: Enter Deepsleep mode by WFE

Example:

```
/* enter deep sleep mode */
pwc_deep_sleep_mode_enter(PWC_DEEP_SLEEP_ENTER_WFI);
```

5.20.10 pwc_voltage_regulate_set function

The table below describes the function pwc_voltage_regulate_set.

Table 498. 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

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

PWC_REGULATOR_EXTRA_LOW_POWER: Voltage regulator extra low-power mode in Deepsleep mode

Example:

```
/* config the voltage regulator mode */
pwc_voltage_regulate_set(PWC_REGULATOR_LOW_POWER);
```

5.20.11 pwc_standby_mode_enter function

The table below describes the function pwc_standby_mode_enter

Table 499. 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();
```

5.20.12 pwc_ldo_output_voltage_set function

The table below describes the function pwc_ldo_output_voltage_set

Table 500. 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:

PWC_LDO_OUTPUT_1V0: LDO output voltage 1.0V

PWC_LDO_OUTPUT_1V2: LDO output voltage 1.2V

Example:

```
/* reduce ldo before enter deepsleep mode */  
pwc_ldo_output_voltage_set(PWC_LDO_OUTPUT_1V0);
```

5.21 Analog-to-digital converter (ADC)

ADC register structure `adc_type` is defined in the “at32l021_adc.h”.

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

The table below gives a list of the ADC registers.

Table 501. 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
vmhb	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 oversample register

The table below gives a list of ADC library functions.

Table 502. Summary of ADC library functions

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

Function name	Description
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
adc_precharge_enable	Enable oversampling of ordinary group

5.21.1 adc_reset function

The table below describes the function adc_reset.

Table 503. adc_reset function

Name	Description
Function name	adc_reset
Function prototype	void adc_reset(adc_type *adc_x)
Function description	Reset all ADC registers to their reset values
Input parameter	adc_x: indicates the selected ADC This parameter can be ADC1.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset()

Example:

<pre>/* deinitialize adc1 */ adc_reset(ADC1);</pre>

5.21.2 adc_enable function

The table below describes the function adc_enable.

Table 504. adc_enable function

Name	Description
Function name	adc_enable
Function prototype	void adc_enable(adc_type *adc_x, confirm_state new_state)
Function description	Enable/disable A/D converter
Input parameter 1	adc_x: indicates the selected ADC This parameter is used to select ADC1.
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:

<pre>/* enable adc1 */</pre>

```
adc_enable(ADC1, TRUE);
```

Note: Calling the function `adc_enable` while the ADC is enabled triggers the conversion of ordinary channels.

5.21.3 `adc_base_default_para_init` function

The table below describes the function `adc_base_default_para_init`.

Table 505. `adc_base_default_para_init` function

Name	Description
Function name	<code>adc_base_default_para_init</code>
Function prototype	<code>void adc_base_default_para_init(adc_base_config_type *adc_base_struct)</code>
Function description	Set the initial value for the <code>adc_base_struct</code> .
Input parameter	<code>adc_base_struct</code> : <code>adc_base_config_type</code> 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);
```

5.21.4 `adc_base_config` function

The table below describes the function `adc_base_config`.

Table 506. `adc_base_config` function

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

`adc_base_config_type` structure

The `adc_base_config_type` is defined in the `at32l021_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);
```

5.21.5 adc_resolution_set function

The table below describes the function `adc_resolution_set`.

Table 507. adc_resolution_set function

Name	Description
Function name	<code>adc_resolution_set</code>
Function prototype	<code>void adc_resolution_set(adc_type *adc_x, adc_resolution_type resolution)</code>
Function description	Set ADC conversion resolution
Input parameter 1	<code>adc_x</code> : indicates the selected ADC peripheral This parameter can be ADC1.
Input parameter 2	<code>new_state</code> : 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

resolution

`resolution` is used to select ADC conversion resolution rate, 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);
```

5.21.6 adc_dma_mode_enable function

The table below describes the function `adc_dma_mode_enable`.

Table 508. adc_dma_mode_enable function

Name	Description
Function name	<code>adc_dma_mode_enable</code>
Function prototype	<code>void adc_dma_mode_enable(adc_type *adc_x, confirm_state new_state)</code>
Function description	Enable DMA transfer for ordinary group conversion
Input parameter 1	<code>adc_x</code> : indicates the selected ADC peripheral This parameter can be ADC1.
Input parameter 2	<code>new_state</code> : 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);
```

5.21.7 adc_interrupt_enable function

The table below describes the function `adc_interrupt_enable`.

Table 509. `adc_interrupt_enable` function

Name	Description
Function name	<code>adc_interrupt_enable</code>
Function prototype	<code>void adc_interrupt_enable(adc_type *adc_x, uint32_t adc_int, confirm_state new_state)</code>
Function description	Enable the selected ADC event interrupt
Input parameter 1	<code>adc_x</code> : indicates the selected ADC peripheral This parameter can be ADC1.
Input parameter 2	<code>adc_int</code> : ADC event interrupt selection This parameter is used to select any event interrupt supported by ADC.
Input parameter3	<code>new_state</code> : indicates the pre-configured status of ADC event interrupts This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

`adc_int`

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

ADC_CCE_INT: Interrupt enabled at the end of channel conversion

ADC_VMOR_INT: Interrupt enabled when voltage monitor is outside a threshold

ADC_PCCE_INT: Interrpt enabled at the end of preempted group conversion

Example:

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

5.21.8 adc_calibration_init function

The table below describes the function `adc_calibration_init`.

Table 510. `adc_calibration_init` function

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

Example:

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

5.21.9 adc_calibration_init_status_get function

The table below describes the function `adc_calibration_init_status_get`.

Table 511. adc_calibration_init_status_get function

Name	Description
Function name	<code>adc_calibration_init_status_get</code>
Function prototype	<code>flag_status adc_calibration_init_status_get(adc_type *adc_x)</code>
Function description	Get the status of initialization calibration
Input parameter	<code>adc_x</code> : indicates the selected ADC peripheral This parameter can be ADC1.
Output parameter	NA
Return value	<code>flag_status</code> : 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));
```

5.21.10 adc_calibration_start function

The table below describes the function `adc_calibration_start`.

Table 512. adc_calibration_start function

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

Example:

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

5.21.11 adc_calibration_status_get function

The table below describes the function `adc_calibration_status_get`.

Table 513. adc_calibration_status_get function

Name	Description
Function name	<code>adc_calibration_status_get</code>
Function prototype	<code>flag_status adc_calibration_status_get(adc_type *adc_x)</code>
Function description	Get the status of calibration
Input parameter	<code>adc_x</code> : indicates the selected ADC peripheral This parameter can be ADC1.
Output parameter	NA
Return value	<code>flag_status</code> : 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));
```

5.21.12 adc_voltage_monitor_enable function

The table below describes the function `adc_voltage_monitor_enable`.

Table 514. adc_voltage_monitor_enable function

Name	Description
Function name	<code>adc_voltage_monitor_enable</code>
Function prototype	<code>void adc_voltage_monitor_enable(adc_type *adc_x, adc_voltage_monitoring_type adc_voltage_monitoring)</code>
Function description	Enable voltage monitor for ordinary/preempted group and for a single channel
Input parameter 1	<code>adc_x</code> : indicates the selected ADC peripheral This parameter can be ADC1.
Input parameter 2	<code>adc_voltage_monitoring</code> : select ordinary group, preempted group or a single channel for voltage monitoring This parameter can be any enumerated value in the <code>adc_voltage_monitoring_type</code> .
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);
```

5.21.13 adc_voltage_monitor_threshold_value_set function

The table below describes the function `adc_voltage_monitor_threshold_value_set`.

Table 515. adc_voltage_monitor_threshold_value_set function

Name	Description
Function name	<code>adc_voltage_monitor_threshold_value_set</code>
Function prototype	<code>void adc_voltage_monitor_threshold_value_set(adc_type *adc_x, uint16_t adc_high_threshold, uint16_t adc_low_threshold)</code>
Function description	Configure the threshold of voltage monitoring
Input parameter 1	<code>adc_x</code> : indicates the selected ADC peripheral This parameter can be ADC1.
Input parameter 2	<code>adc_high_threshold</code> : indicates the upper limit for voltage monitoring This parameter can be any value between 0x000 and 0xFFFF.
Input parameter3	<code>adc_low_threshold</code> : indicates the lower limit for voltage monitoring This parameter can be any value lower than that of <code>adc_high_threshold</code> 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);
```

5.21.14 adc_voltage_monitor_single_channel_select function

The table below describes the function `adc_voltage_monitor_single_channel_select`.

Table 516. `adc_voltage_monitor_single_channel_select` function

Name	Description
Function name	<code>adc_voltage_monitor_single_channel_select</code>
Function prototype	<code>void adc_voltage_monitor_single_channel_select(adc_type *adc_x, adc_channel_select_type adc_channel)</code>
Function description	Select a single channel for voltage monitoring
Input parameter 1	<code>adc_x</code> : indicates the selected ADC peripheral This parameter can be ADC1.
Input parameter 2	<code>adc_channel</code> : 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_16`: ADC channel 16

`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);
```

5.21.15 adc_ordinary_channel_set function

The table below describes the function `adc_ordinary_channel_set`.

Table 517. adc_ordinary_channel_set function

Name	Description
Function name	<code>adc_ordinary_channel_set</code>
Function prototype	<code>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)</code>
Function description	Configure ordinary channels, including parameters such as channel selection, conversion sequence number and sampling time
Input parameter 1	<code>adc_x</code> : indicates the selected ADC peripheral This parameter can be ADC1.
Input parameter 2	<code>adc_channel</code> : indicates the selected channel Refer to adc_channel for details.
Input parameter3	<code>adc_sequence</code> : defines the sequence of channel conversion This parameter can be any value from 1 to 16.
Input parameter4	<code>adc_sampletime</code> : defines the sampling time for channel Refer to adc_sampletime for details.
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_1_5`: sampling time is 1.5 ADCCLK cycles

`ADC_SAMPLETIME_7_5`: sampling time is 7.5 ADCCLK cycles

`ADC_SAMPLETIME_13_5`: sampling time is 13.5 ADCCLK cycles

`ADC_SAMPLETIME_28_5`: sampling time is 28.5 ADCCLK cycles

`ADC_SAMPLETIME_41_5`: sampling time is 41.5 ADCCLK cycles

`ADC_SAMPLETIME_55_5`: sampling time is 55.5 ADCCLK cycles

`ADC_SAMPLETIME_71_5`: sampling time is 71.5 ADCCLK cycles

`ADC_SAMPLETIME_239_5`: sampling time is 239.5 ADCCLK cycles

Example:

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


5.21.16 adc_preempt_channel_length_set function

The table below describes the function `adc_preempt_channel_length_set`.

Table 518. adc_preempt_channel_length_set function

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

Example:

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

5.21.17 adc_preempt_channel_set function

The table below describes the function `adc_preempt_channel_set`.

Table 519. adc_preempt_channel_set function

Name	Description
Function name	<code>adc_preempt_channel_set</code>
Function prototype	<code>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)</code>
Function description	Configure preempted group, including parameters such as channel selection, conversion sequence number and sampling time
Input parameter 1	<code>adc_x</code> : indicates the selected ADC This parameter can be ADC1.
Input parameter 2	<code>adc_channel</code> : indicates the selected channel Refer to adc_channel for details.
Input parameter3	<code>adc_sequence</code> : set the sequence number for channel conversion This parameter can be any value from 1 to 4.
Input parameter4	<code>adc_sampletime</code> : 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_239_5);
adc_preempt_channel_set(ADC1, ADC_CHANNEL_8, 2, ADC_SAMPLETIME_239_5);
```

5.21.18 adc_ordinary_conversion_trigger_set function

The table below describes the function `adc_ordinary_conversion_trigger_set`.

Table 520. adc_ordinary_conversion_trigger_set function

Name	Description
Function name	<code>adc_ordinary_conversion_trigger_set</code>
Function prototype	<code>void adc_ordinary_conversion_trigger_set(adc_type *adc_x, adc_ordinary_trig_select_type adc_ordinary_trig, confirm_state new_state)</code>
Function description	Enable trigger mode and select trigger events for ordinary group conversion
Input parameter 1	<code>adc_x</code> : indicates the selected ADC This parameter can be ADC1.
Input parameter 2	<code>adc_ordinary_trig</code> : indicates the selected trigger event for ordinary group This parameter can be any enumerated value in the <code>adc_ordinary_trig_select_type</code> .
Input parameter3	<code>new_state</code> : indicates the pre-configured status of trigger mode This parameter can be TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

adc_ordinary_trig

The `adc_ordinary_trig` is used to select a trigger event for ordinary group conversion, including:

ADC12_ORDINARY_TRIG_TMR1CH1: TMR1 CH1 event

ADC12_ORDINARY_TRIG_TMR1CH2: TMR1 CH2 event

ADC12_ORDINARY_TRIG_TMR1CH3: TMR1 CH3 event

ADC12_ORDINARY_TRIG_TMR3TRGOUT: TMR3 TRGOUT event

ADC12_ORDINARY_TRIG_TMR15CH1: TMR15 CH1 event

ADC12_ORDINARY_TRIG_EXINT11_TMR8TRGOUT: EXINT 11/TMR8 TRGOUT event

ADC12_ORDINARY_TRIG_SOFTWARE: Software trigger event

Example:

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

5.21.19 adc_preempt_conversion_trigger_set function

The table below describes the function `adc_preempt_conversion_trigger_set`.

Table 521. adc_preempt_conversion_trigger_set function

Name	Description
Function name	<code>adc_preempt_conversion_trigger_set</code>
Function prototype	<code>void adc_preempt_conversion_trigger_set(adc_type *adc_x, adc_preempt_trig_select_type adc_preempt_trig, confirm_state new_state)</code>
Function description	Enable trigger mode and trigger events for preempted group conversion
Input parameter 1	<code>adc_x</code> : indicates the selected ADC This parameter can be ADC1.
Input parameter 2	<code>adc_preempt_trig</code> : indicates the selected trigger event for preempted group This parameter can be any enumerated value in the <code>adc_preempt_trig_select_type</code> .
Input parameter3	<code>new_state</code> : indicates the pre-configured status of trigger mode This parameter can be TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

adc_preempt_trig

The `adc_preempt_trig` is used to select a trigger event for preempted group conversion, including:

`ADC12_PREEMPT_TRIG_TMR1TRGOUT`: TMR1 TRGOUT event
`ADC12_PREEMPT_TRIG_TMR1CH4`: TMR1 CH4 event
`ADC12_PREEMPT_TRIG_TMR3CH4`: TMR3 CH4 event
`ADC12_PREEMPT_TRIG_TMR15TRGOUT`: TMR15 TRGOUT event
`ADC12_PREEMPT_TRIG_EXINT15_TMR1CH4`: EXINT15/TMR1 CH4 event
`ADC12_PREEMPT_TRIG_SOFTWARE`: Software trigger event

Example:

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

5.21.20 adc_preempt_offset_value_set function

The table below describes the function `adc_preempt_offset_value_set`.

Table 522. adc_preempt_offset_value_set function

Name	Description
Function name	<code>adc_preempt_offset_value_set</code>
Function prototype	<code>void adc_preempt_offset_value_set(adc_type *adc_x, adc_preempt_channel_type adc_preempt_channel, uint16_t adc_offset_value)</code>
Function description	Set the offset value of preempted group conversion
Input parameter 1	<code>adc_x</code> : selected ADC This parameter can be ADC1.
Input parameter 2	<code>adc_preempt_channel</code> : indicates the selected channel Refer to adc_preempt_channel for details.
Input parameter3	<code>adc_offset_value</code> : set the offset value for the selected channel This parameter can be any value from 0x000 to 0xFFFF.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

adc_preempt_channel

The `adc_preempt_channel` is used to set an offset value for the selected channel, including:

<code>ADC_PREEMPT_CHANNEL_1</code> :	Preempted channel 1
<code>ADC_PREEMPT_CHANNEL_2</code> :	Preempted channel 2
<code>ADC_PREEMPT_CHANNEL_3</code> :	Preempted channel 3
<code>ADC_PREEMPT_CHANNEL_4</code> :	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);
```

5.21.21 adc_ordinary_part_count_set function

The table below describes the function `adc_ordinary_part_count_set`.

Table 523. adc_ordinary_part_count_set function

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

Example:

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

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

5.21.22 adc_ordinary_part_mode_enable function

The table below describes the function `adc_ordinary_part_mode_enable`.

Table 524. adc_ordinary_part_mode_enable function

Name	Description
Function name	<code>adc_ordinary_part_mode_enable</code>
Function prototype	<code>void adc_ordinary_part_mode_enable(adc_type *adc_x, confirm_state new_state)</code>
Function description	Enable partition mode for ordinary channels
Input parameter 1	<code>adc_x</code> : indicates the selected ADC This parameter can be ADC1.
Input parameter 2	<code>new_state</code> : 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);
```

5.21.23 adc_preempt_part_mode_enable function

The table below describes the function `adc_preempt_part_mode_enable`.

Table 525. adc_preempt_part_mode_enable function

Name	Description
Function name	<code>adc_preempt_part_mode_enable</code>
Function prototype	<code>void adc_preempt_part_mode_enable(adc_type *adc_x, confirm_state new_state)</code>
Function description	Enable partition mode for preempted channels
Input parameter 1	<code>adc_x</code> : indicates the selected ADC This parameter can be ADC1.
Input parameter 2	<code>new_state</code> : 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);
```

5.21.24 adc_preempt_auto_mode_enable function

The table below describes the function `adc_preempt_auto_mode_enable`.

Table 526. adc_preempt_auto_mode_enable function

Name	Description
Function name	<code>adc_preempt_auto_mode_enable</code>
Function prototype	<code>void adc_preempt_auto_mode_enable(adc_type *adc_x, confirm_state, new_state)</code>
Function description	Enable auto preempted group conversion at the end of ordinary group conversion
Input parameter 1	<code>adc_x</code> : indicates the selected ADC This parameter can be ADC1.
Input parameter 2	<code>new_state</code> : 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);
```

5.21.25 adc_tempersensor_vintrv_enable function

The table below describes the function `adc_tempersensor_vintrv_enable`.

Table 527. adc_tempersensor_vintrv_enable

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

Example:

```
/* enable the temperature sensor and vintrv channel */
adc_tempersensor_vintrv_enable(TRUE);
```

5.21.26 adc_ordinary_software_trigger_enable function

The table below describes the function `adc_ordinary_software_trigger_enable`.

Table 528. adc_ordinary_software_trigger_enable function

Name	Description
Function name	<code>adc_ordinary_software_trigger_enable</code>
Function prototype	<code>void adc_ordinary_software_trigger_enable(adc_type *adc_x, confirm_state new_state)</code>
Function description	Trigger ordinary group conversion by software
Input parameter 1	<code>adc_x</code> : indicates the selected ADC This parameter can be ADC1.
Input parameter 2	<code>new_state</code> : 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);
```


5.21.27 adc_ordinary_software_trigger_status_get function

The table below describes the function `adc_ordinary_software_trigger_status_get`

Table 529. adc_ordinary_software_trigger_status_get function

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

Example:

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

5.21.28 adc_preempt_software_trigger_enable function

The table below describes the function `adc_preempt_software_trigger_enable`

Table 530. adc_preempt_software_trigger_enable function

Name	Description
Function name	<code>adc_preempt_software_trigger_enable</code>
Function prototype	<code>void adc_preempt_software_trigger_enable(adc_type *adc_x, confirm_state new_state)</code>
Function description	Preempted group conversion triggered by software
Input parameter 1	<code>adc_x</code> : indicates the selected ADC This parameter can be ADC1.
Input parameter 2	<code>new_state</code> : 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);
```

5.21.29 adc_preempt_software_trigger_status_get function

The table below describes the function `adc_preempt_software_trigger_status_get`.

Table 531. adc_preempt_software_trigger_status_get function

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

Example:

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

5.21.30 adc_ordinary_conversion_data_get function

The table below describes the function `adc_ordinary_conversion_data_get`.

Table 532. adc_ordinary_conversion_data_get function

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

Example:

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

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

5.21.31 adc_preempt_conversion_data_get function

The table below describes the function `adc_preempt_conversion_data_get`.

Table 533. `adc_preempt_conversion_data_get` function

Name	Description
Function name	<code>adc_preempt_conversion_data_get</code>
Function prototype	<code>uint16_t adc_preempt_conversion_data_get(adc_type *adc_x, adc_preempt_channel_type adc_preempt_channel)</code>
Function description	Get the converted data of preempted group
Input parameter 1	<code>adc_x</code> : indicates the selected ADC This parameter can be ADC1.
Input parameter 2	<code>adc_preempt_channel</code> : 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);
```

5.21.32 adc_flag_get function

The table below describes the function `adc_flag_get`.

Table 534. adc_flag_get function

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

adc_flag

The `adc_flag` is used to select a flag to get its status, including:

`ADC_VMOR_FLAG`: Voltage monitor outside threshold
`ADC_CCE_FLAG`: End of channel conversion
`ADC_PCCE_FLAG`: End of preempted group conversion
`ADC_PCCS_FLAG`: Start of preempted channel conversion
`ADC_OCCS_FLAG`: Start of ordinary channel conversion
`ADC_RDY_FLAG`: ADC ready flag

Example:

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

5.21.33 adc_interrupt_flag_get function

The table below describes the function `adc_interrupt_flag_get`.

Table 535. adc_interrupt_flag_get function

Name	Description
Function name	<code>adc_interrupt_flag_get</code>
Function prototype	<code>flag_status adc_interrupt_flag_get(adc_type *adc_x, uint8_t adc_flag)</code>
Function description	Get interrupt flag status
Input parameter 1	<code>adc_x</code> : indicates the selected ADC This parameter can be ADC1.
Input parameter 2	<code>adc_flag</code> : select a flag to be clear. Refer to adc_flag
Output parameter	NA
Return value	<code>flag_status</code> : flag status 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_CCE_FLAG: End of channel conversion

ADC_PCCE_FLAG: End of preempted group conversion

Example:

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

5.21.34 adc_flag_clear function

The table below describes the function `adc_flag_clear`.

Table 536. adc_flag_clear function

Name	Description
Function name	<code>adc_flag_clear</code>
Function prototype	<code>void adc_flag_clear(adc_type *adc_x, uint32_t adc_flag)</code>
Function description	Clear the flag bits that have been set.
Input parameter 1	<code>adc_x</code> : indicates the selected ADC This parameter can be ADC1.
Input parameter 2	<code>adc_flag</code> : select a flag to be clear. Refer to adc_flag
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);
```

5.21.35 adc_ordinary_oversample_enable function

The table below describes the function `adc_ordinary_oversample_enable`.

Table 537. `adc_ordinary_oversample_enable` function

Name	Description
Function name	<code>adc_ordinary_oversample_enable</code>
Function prototype	<code>void adc_ordinary_oversample_enable(adc_type *adc_x, confirm_state new_state)</code>
Function description	Enable oversampling for ordinary group
Input parameter 1	<code>adc_x</code> : indicates the selected ADC This parameter can be ADC1.
Input parameter 2	<code>new_state</code> : 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
Called functions	NA

Example:

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

5.21.36 adc_preempt_oversample_enable function

The table below describes the function `adc_preempt_oversample_enable`.

Table 538. adc_preempt_oversample_enable function

Name	Description
Function name	<code>adc_preempt_oversample_enable</code>
Function prototype	<code>void adc_preempt_oversample_enable(adc_type *adc_x, confirm_state new_state)</code>
Function description	Enable oversampling for preempted group
Input parameter 1	<code>adc_x</code> : indicates the selected ADC This parameter can be ADC1.
Input parameter 2	<code>new_state</code> : 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);
```

5.21.37 adc_oversample_ratio_shift_set function

The table below describes the function `adc_oversample_ratio_shift_set`.

Table 539. adc_oversample_ratio_shift_set function

Name	Description
Function name	<code>adc_oversample_ratio_shift_set</code>
Function prototype	<code>void adc_oversample_ratio_shift_set(adc_type *adc_x, adc_oversample_ratio_type adc_oversample_ratio, adc_oversample_shift_type adc_oversample_shift)</code>
Function description	Set oversampling ratio and shift mode
Input parameter 1	<code>adc_x</code> : indicates the selected ADC This parameter can be ADC1.
Input parameter 2	<code>adc_oversample_ratio</code> : indicates the oversampling rate This parameter can be any enumerated value in the <code>adc_oversample_ratio_type</code> .
Input parameter3	<code>adc_oversample_shift</code> : indicates the oversampling shift mode This parameter can be any enumerated value in the <code>adc_oversample_shift_type</code> .
Output parameter	NA
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);

```

5.21.38 adc_ordinary_oversample_trig_enable function

The table below describes the function adc_ordinary_oversample_trig_enable.

Table 540. 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.
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
Required preconditions	NA
Called functions	NA

Example:

```

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

```


5.21.39 adc_ordinary_oversample_restart_set function

The table below describes the function `adc_ordinary_oversample_restart_set`.

Table 541. `adc_ordinary_oversample_restart_set` function

Name	Description
Function name	<code>adc_ordinary_oversample_restart_set</code>
Function prototype	<code>void adc_ordinary_oversample_restart_set(adc_type *adc_x, adc_ordinary_oversample_restart_type adc_ordinary_oversample_restart)</code>
Function description	Select oversampling restart mode for ordinary group
Input parameter 1	<code>adc_x</code> : indicates the selected ADC This parameter can be ADC1.
Input parameter 2	<code>adc_ordinary_oversample_restart</code> : indicates the pre-configured status of oversampling restart mode for ordinary group This parameter can be any enumerated value in the <code>adc_ordinary_oversample_restart_type</code> .
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);
```

5.21.40 adc_precharge_enable function

The table below describes the function adc_precharge_enable.

Table 542. adc_precharge_enable function

Name	Description
Function name	adc_precharge_enable
Function prototype	void adc_precharge_enable(adc_type *adc_x, confirm_state new_state)
Function description	Enable precharge
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1.
Input parameter 2	new_state: precharge enable state This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
/* precharge enable.precharge function need be disable when calibration */
adc_precharge_enable(ADC1, TRUE);
```

6 Revision history

Table 543. Document revision history

Date	Revision	Changes
2024.01.11	2.0.0	Initial release
2024.03.01	2.0.1	Added the 5.20.12 pwc_ldo_output_voltage_set 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.

© 2024 Artery Technology -All rights reserved