

GigaDevice Semiconductor Inc.

GD32F4xx
ARM® Cortex™-M4 32-bit MCU

Firmware Library
User Guide

Revision 1.2

(Dec. 2022)

Table of Contents

Table of Contents	1
List of Figures	4
List of Tables	5
1. Introduction.....	27
1.1. Rules of User Manual and Firmware Library	27
1.1.1. Peripherals.....	27
1.1.2. Naming rules.....	28
2. Firmware Library Overview	30
2.1. File Structure of Firmware Library	30
2.1.1. Examples Folder	31
2.1.2. Firmware Folder.....	32
2.1.3. Project Folder.....	32
2.1.4. Template Folder	32
2.1.5. Utilities Folder	35
2.2. File descriptions of Firmware Library	36
3. Firmware Library of Standard Peripherals	37
3.1. Overview of Firmware Library of Standard Peripherals.....	37
3.2. ADC	37
3.2.1. Descriptions of Peripheral registers.....	37
3.2.2. Descriptions of Peripheral functions	38
3.3. CAN	81
3.3.1. Descriptions of Peripheral registers.....	81
3.3.2. Descriptions of Peripheral functions	82
3.4. CRC	115
3.4.1. Descriptions of Peripheral registers.....	116
3.4.2. Descriptions of Peripheral functions	116
3.5. CTC.....	120
3.5.1. Descriptions of Peripheral registers.....	120
3.5.2. Descriptions of Peripheral functions	120
3.6. DAC	137
3.6.1. Descriptions of Peripheral registers.....	137
3.6.2. Descriptions of Peripheral functions	137
3.7. DBG	156
3.7.1. Descriptions of Peripheral registers.....	156

3.7.2. Descriptions of Peripheral functions	157
3.8. DCI	163
3.8.1. Descriptions of Peripheral registers	163
3.8.2. Descriptions of Peripheral functions	164
3.9. DMA	180
3.9.1. Descriptions of Peripheral registers	180
3.9.2. Descriptions of Peripheral functions	181
3.10. ENET	211
3.10.1. Descriptions of Peripheral registers	212
3.10.2. Descriptions of Peripheral functions	215
3.11. EXMC	308
3.11.1. Descriptions of Peripheral registers	308
3.11.2. Descriptions of Peripheral functions	309
3.12. EXTI	345
3.12.1. Descriptions of Peripheral registers	345
3.12.2. Descriptions of Peripheral functions	345
3.13. FMC	353
3.13.1. Descriptions of Peripheral registers	353
3.13.2. Descriptions of Peripheral functions	353
3.14. FWDGT	377
3.14.1. Descriptions of Peripheral registers	377
3.14.2. Descriptions of Peripheral functions	378
3.15. GPIO	383
3.15.1. Descriptions of Peripheral registers	384
3.15.2. Descriptions of Peripheral functions	384
3.16. I2C	395
3.16.1. Descriptions of Peripheral registers	395
3.16.2. Descriptions of Peripheral functions	395
3.17. IPA	428
3.17.1. Descriptions of Peripheral registers	428
3.17.2. Descriptions of Peripheral functions	429
3.18. IREF	452
3.18.1. Descriptions of Peripheral registers	452
3.18.2. Descriptions of Peripheral functions	452
3.19. MISC	456
3.19.1. Descriptions of Peripheral registers	456
3.19.2. Descriptions of Peripheral functions	457
3.20. PMU	464
3.20.1. Descriptions of Peripheral registers	464

3.20.2. Descriptions of Peripheral functions	465
3.21. RCU	479
3.21.1. Descriptions of Peripheral registers	480
3.21.2. Descriptions of Peripheral functions	481
3.22. RTC	521
3.22.1. Descriptions of Peripheral registers	521
3.22.2. Descriptions of Peripheral functions	522
3.23. SDIO	556
3.23.1. Descriptions of Peripheral registers	557
3.23.2. Descriptions of Peripheral functions	557
3.24. SPI	596
3.24.1. Descriptions of Peripheral registers	596
3.24.2. Descriptions of Peripheral functions	597
3.25. SYSCFG	624
3.25.1. Descriptions of Peripheral registers	624
3.25.2. Descriptions of Peripheral functions	624
3.26. TIMER.....	630
3.26.1. Descriptions of Peripheral registers	631
3.26.2. Descriptions of Peripheral functions	632
3.27. TLI	704
3.27.1. Descriptions of Peripheral registers	704
3.27.2. Descriptions of Peripheral functions	706
3.28. TRNG.....	728
3.28.1. Descriptions of Peripheral registers	729
3.28.2. Descriptions of Peripheral functions	729
3.29. USART.....	735
3.29.1. Descriptions of Peripheral registers	735
3.29.2. Descriptions of Peripheral functions	736
3.30. WWDGT.....	772
3.30.1. Descriptions of Peripheral registers	772
3.30.2. Descriptions of Peripheral functions	772
4. Revision history	779

List of Figures

Figure 2-1. File structure of firmware library of GD32F4xx	31
Figure 2-2. Select peripheral example files	33
Figure 2-3. Copy the peripheral example files	34
Figure 2-4. Open the project file	34
Figure 2-5. Configure project files	35
Figure 2-6. Compile-debug-download	35

List of Tables

Table 1-1. Peripherals	27
Table 2-1. Function descriptions of Firmware Library.....	36
Table 3-1. Peripheral function format of Firmware Library	37
Table 3-2. ADC Registers	37
Table 3-3. ADC firmware function.....	38
Table 3-4. Function adc_deinit.....	39
Table 3-5. Function adc_clock_config	40
Table 3-6. Function adc_special_function_config.....	41
Table 3-7. Function adc_data_alignment_config.....	42
Table 3-8. Function adc_enable	43
Table 3-9. Function adc_disable	44
Table 3-10. Function adc_calibration_enable.....	44
Table 3-11. Function adc_channel_16_to_18	45
Table 3-12. Function adc_resolution_config	46
Table 3-13. Function adc_oversample_mode_config	47
Table 3-14. Function adc_oversample_mode_enable	49
Table 3-15. Function adc_oversample_mode_disable	49
Table 3-16. Function adc_dma_mode_enable.....	50
Table 3-17. Function adc_dma_mode_disable.....	51
Table 3-18. Function adc_dma_request_after_last_enable	51
Table 3-19. Function adc_dma_mode_disable.....	52
Table 3-20. Function adc_discontinuous_mode_config	53
Table 3-21. Function adc_channel_length_config.....	54
Table 3-22. Function adc_regular_channel_config	55
Table 3-23. Function adc_inserted_channel_config	56
Table 3-24. Function adc_inserted_channel_offset_config	57
Table 3-25. Function adc_external_trigger_source_config	58
Table 3-26. Function adc_external_trigger_config.....	61
Table 3-27. Function adc_software_trigger_enable	62
Table 3-28. Function adc_end_of_conversion_config.....	63
Table 3-29. Function adc_regular_data_read	64
Table 3-30. Function adc_inserted_data_read	65
Table 3-31. Function adc_watchdog_single_channel_disable.....	66
Table 3-32. Function adc_watchdog_single_channel_enable.....	66
Table 3-33. Function adc_watchdog_group_channel_enable	67
Table 3-34. Function adc_watchdog_disable	68
Table 3-35. Function adc_watchdog_threshold_config	69
Table 3-36. Function adc_flag_get	69
Table 3-37. Function adc_flag_clear	70
Table 3-38. Function adc_regular_software_startconv_flag_get.....	71

Table 3-39. Function adc_inserted_software_startconv_flag_get.....	72
Table 3-40. Function adc_interrupt_flag_get.....	73
Table 3-41. Function adc_interrupt_flag_clear	74
Table 3-42. Function adc_interrupt_enable	74
Table 3-43. Function adc_interrupt_disable	75
Table 3-44. Function adc_sync_mode_config.....	76
Table 3-45. Function adc_interrupt_disable	78
Table 3-46. Function adc_sync_dma_config	78
Table 3-47. Function adc_sync_dma_request_after_last_enable	79
Table 3-48. Function adc_sync_dma_request_after_last_disable	80
Table 3-49. Function adc_interrupt_disable	80
Table 3-50. CAN Registers	81
Table 3-51. CAN firmware function	82
Table 3-52. can_parameter_struct.....	83
Table 3-53. can_trasnmit_message_struct	84
Table 3-54. can_receive_message_struct	84
Table 3-55. can_filter_parameter_struct.....	84
Table 3-56. Function can_deinit.....	85
Table 3-57. Function can_struct_para_init	86
Table 3-58. Function can_init	87
Table 3-59. Function can_filter_init.....	88
Table 3-60. Function can1_filter_start_bank	89
Table 3-61. Function can_debug_freeze_enable	90
Table 3-62. Function can_debug_freeze_disable	91
Table 3-63. Function can_time_trigger_mode_enable.....	92
Table 3-64. Function can_time_trigger_mode_disable.....	93
Table 3-65. Function can_message_transmit.....	94
Table 3-66. Function can_transmit_states	95
Table 3-67. Function can_transmission_stop	97
Table 3-68. Function can_message_receive	98
Table 3-69. Function can_fifo_release.....	100
Table 3-70. Function can_receive_message_length_get	101
Table 3-71. Function can_working_mode_set.....	102
Table 3-72. Function can_wakeup	103
Table 3-73. Function can_error_get	104
Table 3-74. Function can_receive_error_number_get	105
Table 3-75. Function can_transmit_error_number_get	106
Table 3-76. Function can_flag_get	107
Table 3-77. Function can_flag_clear	109
Table 3-78. Function can_interrupt_enable	110
Table 3-79. Function can_interrupt_disable	112
Table 3-80. Function can_interrupt_flag_get.....	113
Table 3-81. Function can_interrupt_flag_clear	114
Table 3-82. CRC Registers	116

Table 3-83. CRC firmware function	116
Table 3-84. Function <code>crc_deinit</code>	116
Table 3-85. Function <code>crc_data_register_reset</code>	117
Table 3-86. Function <code>crc_data_register_read</code>	117
Table 3-87. Function <code>crc_free_data_register_read</code>	118
Table 3-88. Function <code>crc_free_data_register_write</code>	118
Table 3-89. Function <code>crc_single_data_calculate</code>	119
Table 3-90. Function <code>crc_block_data_calculate</code>	119
Table 3-91. CTC Registers	120
Table 3-92. CTC firmware function	120
Table 3-93. Function <code>ctc_deinit</code>	121
Table 3-94. Function <code>ctc_counter_enable</code>	122
Table 3-95. Function <code>ctc_counter_disable</code>	123
Table 3-96. Function <code>ctc_irc48m_trim_value_config</code>	123
Table 3-97. Function <code>ctc_software_refresource_pulse_generate</code>	124
Table 3-98. Function <code>ctc_hardware_trim_mode_config</code>	124
Table 3-99. Function <code>ctc_refresource_polarity_config</code>	125
Table 3-101. Function <code>ctc_refresource_signal_select</code>	126
Table 3-102. Function <code>ctc_refresource_prescaler_config</code>	127
Table 3-103. Function <code>ctc_clock_limit_value_config</code>	128
Table 3-104. Function <code>ctc_counter_reload_value_config</code>	128
Table 3-105. Function <code>ctc_counter_capture_value_read</code>	129
Table 3-106. Function <code>ctc_counter_direction_read</code>	130
Table 3-107. Function <code>ctc_counter_reload_value_read</code>	130
Table 3-108. Function <code>ctc_irc48m_trim_value_read</code>	131
Table 3-109. Function <code>ctc_interrupt_enable</code>	132
Table 3-110. Function <code>ctc_interrupt_disable</code>	132
Table 3-111. Function <code>ctc_interrupt_flag_get</code>	133
Table 3-112. Function <code>ctc_interrupt_flag_clear</code>	134
Table 3-113. Function <code>ctc_flag_get</code>	135
Table 3-114. Function <code>ctc_flag_clear</code>	136
Table 3-115. DAC Registers	137
Table 3-116. DAC firmware function.....	137
Table 3-117. Function <code>dac_deinit</code>	138
Table 3-118. Function <code>dac_enable</code>	139
Table 3-119. Function <code>dac_disable</code>	139
Table 3-120. Function <code>dac_dma_enable</code>	140
Table 3-121. Function <code>dac_dma_disable</code>	140
Table 3-122. Function <code>dac_output_buffer_enable</code>	141
Table 3-123. Function <code>dac_output_buffer_disable</code>	141
Table 3-124. Function <code>dac_output_value_get</code>	142
Table 3-125. Function <code>dac_data_set</code>	142
Table 3-126. Function <code>dac_trigger_enable</code>	143
Table 3-127. Function <code>dac_trigger_disable</code>	144

Table 3-128. Function dac_trigger_source_config.....	144
Table 3-129. Function dac_software_trigger_enable	145
Table 3-130. Function dac_software_trigger_disable	146
Table 3-131. Function dac_wave_mode_config.....	146
Table 3-132. Function dac_wave_bit_width_config	147
Table 3-133. Function dac_lfsr_noise_config	147
Table 3-134. Function dac_triangle_noise_config.....	148
Table 3-135. Function dac_concurrent_enable.....	149
Table 3-136. Function dac_concurrent_disable.....	149
Table 3-137. Function dac_concurrent_software_trigger_enable	150
Table 3-138. Function dac_concurrent_software_trigger_disable	150
Table 3-139. Function dac_concurrent_output_buffer_enable	151
Table 3-140. Function dac_concurrent_output_buffer_disable	151
Table 3-141. Function dac_concurrent_data_set	152
Table 3-142. Function dac_concurrent_interrupt_enable	152
Table 3-143. Function dac_concurrent_interrupt_disable	153
Table 3-144. Function dac_flag_get	153
Table 3-145. Function dac_flag_clear	154
Table 3-146. Function dac_interrupt_enable	154
Table 3-147. Function dac_interrupt_disable	155
Table 3-148. Function dac_interrupt_flag_get.....	155
Table 3-149. Function dac_interrupt_flag_get.....	156
Table 3-150. DBG Registers.....	157
Table 3-151. DBG firmware function	157
Table 3-152. Enum dbg_periph_enum	157
Table 3-153. Function dbg_id_get	158
Table 3-154. Function dbg_low_power_enable.....	159
Table 3-155. Function dbg_low_power_disable.....	159
Table 3-156. Function dbg_periph_enable	160
Table 3-157. Function dbg_periph_disable	161
Table 3-158. Function dbg_trace_pin_enable	162
Table 3-159. Function dbg_trace_pin_disable	162
Table 3-160. Function dbg_trace_pin_mode_set.....	错误!未定义书签。
Table 3-161. DCI Registers.....	163
Table 3-162. DCI firmware function	164
Table 3-163. Structure dci_parameter_struct	165
Table 3-164. Function dci_deinit.....	165
Table 3-165. Function dci_init	166
Table 3-166. Function dci_enable	167
Table 3-167. Function dci_disable	167
Table 3-168. Function dci_capture_enable	168
Table 3-169. Function dci_capture_disable	169
Table 3-170. Function dci_jpeg_enable	169
Table 3-171. Function dci_jpeg_disable	170

Table 3-172. Function dci_crop_window_enable.....	170
Table 3-173. Function dci_crop_window_disable	171
Table 3-174. Function dci_crop_window_config	172
Table 3-175. Function dci_embedded_sync_enable	172
Table 3-176. Function dci_embedded_sync_disable	173
Table 3-177. Function dci_sync_codes_config.....	174
Table 3-178. Function dci_sync_codes_unmask_config	175
Table 3-179. Function dci_data_read	175
Table 3-180. Function dci_flag_get	176
Table 3-181. Function dci_interrupt_enable	177
Table 3-182. Function dci_interrupt_disable	178
Table 3-183. Function dci_interrupt_flag_get.....	178
Table 3-184. Function dci_interrupt_flag_clear	179
Table 3-185. DMA Registers.....	180
Table 3-186. DMA firmware function	181
Table 3-187. Structure dma_multi_data_parameter_struct.....	182
Table 3-188. Structure dma_single_data_parameter_struct.....	183
Table 3-189. Function dma_deinit	183
Table 3-190. Function dma_single_data_mode_init.....	184
Table 3-191. Function dma_multi_data_mode_init.....	185
Table 3-192. Function dma_periph_address_config	186
Table 3-193. Function dma_memory_address_config.....	187
Table 3-194. Function dma_transfer_number_config	188
Table 3-195. Function dma_transfer_number_get.....	189
Table 3-196. Function dma_priority_config	189
Table 3-197. Function dma_memory_burst_beats_config.....	190
Table 3-198. Function dma_periph_burst_beats_config.....	191
Table 3-199. Function dma_memory_width_config	192
Table 3-200. Function dma_periph_width_config	193
Table 3-201. Function dma_memory_address_generation_config	194
Table 3-202. Function dma_peripheral_address_generation_config.....	195
Table 3-203. Function dma_circulation_enable	196
Table 3-204. Function dma_circulation_disable	197
Table 3-205. Function dma_channel_enable	198
Table 3-206. Function dma_channel_disable	199
Table 3-207. Function dma_transfer_direction_config	199
Table 3-208. Function dma_switch_buffer_mode_config	200
Table 3-209. Function dma_using_memory_get.....	201
Table 3-210. Function dma_channel_subperipheral_select	202
Table 3-211. Function dma_flow_controller_config	203
Table 3-212. Function dma_switch_buffer_mode_enable.....	204
Table 3-213. Function dma_fifo_status_get.....	205
Table 3-214. Function dma_flag_get	205
Table 3-215. Function dma_flag_clear.....	206

Table 3-216. Function <code>dma_interrupt_flag_get</code>	207
Table 3-217. Function <code>dma_interrupt_flag_clear</code>	208
Table 3-218. Function <code>dma_interrupt_enable</code>	209
Table 3-219. Function <code>dma_interrupt_disable</code>	210
Table 3-220. ENET Registers	212
Table 3-221. ENET firmware function	215
Table 3-222. Structure <code>enet_initpara_struct</code>	219
Table 3-223. Structure <code>enet_descriptors_struct</code>	220
Table 3-224. Structure <code>enet_ptp_systime_struct</code>	220
Table 3-225. Function <code>enet_deinit</code>	221
Table 3-226. Function <code>enet_initpara_config</code>	221
Table 3-227. Function <code>enet_init</code>	226
Table 3-228. Function <code>enet_software_reset</code>	227
Table 3-229. Function <code>enet_rxframe_size_get</code>	228
Table 3-230. Function <code>enet_descriptors_chain_init</code>	229
Table 3-231. Function <code>enet_descriptors_ring_init</code>	229
Table 3-232. Function <code>enet_frame_receive</code>	230
Table 3-233. Function <code>enet_frame_transmit</code>	231
Table 3-234. Function <code>enet_transmit_checksum_config</code>	231
Table 3-235. Function <code>enet_enable</code>	232
Table 3-236. Function <code>enet_disable</code>	233
Table 3-237. Function <code>enet_mac_address_set</code>	234
Table 3-238. Function <code>enet_mac_address_get</code>	235
Table 3-239. Function <code>enet_flag_get</code>	236
Table 3-240. Function <code>enet_flag_clear</code>	238
Table 3-241. Function <code>enet_interrupt_enable</code>	240
Table 3-242. Function <code>enet_interrupt_disable</code>	242
Table 3-243. Function <code>enet_interrupt_flag_get</code>	244
Table 3-244. Function <code>enet_interrupt_flag_clear</code>	246
Table 3-245. Function <code>enet_tx_enable</code>	247
Table 3-246. Function <code>enet_tx_disable</code>	248
Table 3-247. Function <code>enet_rx_enable</code>	249
Table 3-248. Function <code>enet_rx_disable</code>	249
Table 3-249. Function <code>enet_registers_get</code>	250
Table 3-250. Function <code>enet_debug_status_get</code>	251
Table 3-251. Function <code>enet_address_filter_enable</code>	252
Table 3-252. Function <code>enet_address_filter_disable</code>	253
Table 3-253. Function <code>enet_address_filter_config</code>	254
Table 3-254. Function <code>enet_phy_config</code>	255
Table 3-255. Function <code>enet_phy_write_read</code>	256
Table 3-256. Function <code>enet_phyloopback_enable</code>	257
Table 3-257. Function <code>enet_phyloopback_disable</code>	258
Table 3-258. Function <code>enet_forward_feature_enable</code>	258
Table 3-259. Function <code>enet_forward_feature_disable</code>	259

Table 3-260. Function enet_fliter_feature_enable	260
Table 3-261. Function enet_fliter_feature_disable	261
Table 3-262. Function enet_pauseframe_generate	262
Table 3-263. Function enet_pauseframe_detect_config	263
Table 3-264. Function enet_pauseframe_config.....	264
Table 3-265. Function enet_flowcontrol_threshold_config	265
Table 3-266. Function enet_flowcontrol_feature_enable	266
Table 3-267. Function enet_flowcontrol_feature_disable	267
Table 3-268. Function enet_dmaprocess_state_get	268
Table 3-269. Function enet_dmaprocess_resume.....	269
Table 3-270. Function enet_rxprocess_check_recovery.....	270
Table 3-271. Function enet_txfifo_flush	270
Table 3-272. Function enet_current_desc_address_get	271
Table 3-273. Function enet_desc_information_get	272
Table 3-274. Function enet_missed_frame_counter_get	273
Table 3-275. Function enet_desc_flag_get	274
Table 3-276. Function enet_desc_flag_set	276
Table 3-277. Function enet_desc_flag_clear	278
Table 3-278. Function enet_rx_desc_immediate_receive_complete_interrupt.....	279
Table 3-279. Function enet_rx_desc_delay_receive_complete_interrupt.....	280
Table 3-280. Function enet_rxframe_drop	280
Table 3-281. Function enet_dma_feature_enable	281
Table 3-282. Function enet_dma_feature_disable	282
Table 3-283. Function enet_rx_desc_enhanced_status_get	283
Table 3-284. Function enet_desc_select_enhanced_mode	284
Table 3-285. Function enet_ptp_enhanced_descriptors_chain_init	284
Table 3-286. Function enet_ptp_enhanced_descriptors_ring_init	285
Table 3-287. Function enet_ptpframe_receive_enhanced_mode	286
Table 3-288. Function enet_ptpframe_transmit_enhanced_mode	287
Table 3-289. Function enet_desc_select_normal_mode	288
Table 3-290. Function enet_ptp_normal_descriptors_chain_init	288
Table 3-291. Function enet_ptp_normal_descriptors_ring_init	289
Table 3-292. Function enet_ptpframe_receive_normal_mode	290
Table 3-293. Function enet_ptpframe_transmit_normal_mode	291
Table 3-294. Function enet_wum_filter_register_pointer_reset	292
Table 3-295. Function enet_wum_filter_config	292
Table 3-296. Function enet_wum_feature_enable	293
Table 3-297. Function enet_wum_feature_disable	294
Table 3-298. Function enet_msc_counters_reset.....	295
Table 3-299. Function enet_msc_feature_enable	295
Table 3-300. Function enet_msc_feature_disable	296
Table 3-301. Function enet_msc_counters_preset_config.....	297
Table 3-302. Function enet_msc_counters_get.....	298
Table 3-303. Function enet_ptp_feature_enable	299

Table 3-304. Function enet_ptp_feature_disable.....	300
Table 3-305. Function enet_ptp_timestamp_function_config	301
Table 3-306. Function enet_ptp_subsecond_increment_config.....	303
Table 3-307. Function enet_ptp_timestamp_addend_config	303
Table 3-308. Function enet_ptp_timestamp_update_config.....	304
Table 3-309. Function enet_ptp_expected_time_config.....	305
Table 3-310. Function enet_ptp_system_time_get.....	305
Table 3-311. Function enet_ptp_pps_output_frequency_config	306
Table 3-312. Function enet_initpara_reset.....	308
Table 3-313. EXMC Registers	308
Table 3-314. EXMC firmware function.....	309
Table 3-315. Structure exmc_norsram_timing_parameter_struct	310
Table 3-316. Structure exmc_norsram_parameter_struct.....	311
Table 3-317. Structure exmc_nand_pccard_timing_parameter_struct.....	311
Table 3-318. Structure exmc_nand_parameter_struct.....	311
Table 3-319. Structure exmc_pccard_parameter_struct	312
Table 3-320. Structure exmc_sdram_timing_parameter_struct.....	312
Table 3-321. Structure exmc_sdram_parameter_struct	313
Table 3-322. Structure exmc_sdram_command_parameter_struct	313
Table 3-323. Structure exmc_sqipssram_parameter_struct.....	313
Table 3-324. Function exmc_norsram_deinit	313
Table 3-325. Function exmc_norsram_struct_para_init.....	314
Table 3-326. Function exmc_norsram_init.....	315
Table 3-327. Function exmc_norsram_enable	316
Table 3-328. Function exmc_norsram_disable	317
Table 3-329. Function exmc_nand_deinit	317
Table 3-330. Function exmc_nand_struct_para_init	318
Table 3-331. Function exmc_nand_init.....	318
Table 3-332. Function exmc_nand_enable	319
Table 3-333. Function exmc_nand_disable	320
Table 3-334. Function exmc_pccard_deinit	320
Table 3-335. Function exmc_pccard_struct_para_init	321
Table 3-336. Function exmc_pccard_init	321
Table 3-337. Function exmc_pccard_enable	322
Table 3-338. Function exmc_pccard_disable	323
Table 3-339. Function exmc_sdram_deinit	323
Table 3-340. Function exmc_sdram_struct_para_init	324
Table 3-341. Function exmc_sdram_init	324
Table 3-369. Function exmc_sdram_struct_command_para_init.....	326
Table 3-342. Function exmc_sqipssram_deinit	326
Table 3-343. Function exmc_sqipssram_struct_para_init	327
Table 3-344. Function exmc_sqipssram_init	327
Table 3-345. Function exmc_norsram_consecutive_clock_config	328
Table 3-346. Function exmc_norsram_page_size_config	329

Table 3-347. Function exmc_nand_ecc_config.....	330
Table 3-348. Function exmc_ecc_get	330
Table 3-349. Function exmc_sdram_readsampel_enable	331
Table 3-350. Function exmc_sdram_readsampel_config.....	331
Table 3-351. Function exmc_sdram_command_config.....	332
Table 3-352. Function exmc_sdram_refresh_count_set	333
Table 3-353. Function exmc_sdram_autorefresh_number_set.....	333
Table 3-354. Function exmc_sdram_write_protection_config	334
Table 3-355. Function exmc_sdram_bankstatus_get	334
Table 3-356. Function exmc_sqipssram_read_command_set.....	335
Table 3-357. Function exmc_sqipssram_write_command_set.....	336
Table 3-358. Function exmc_sqipssram_read_id_command_send.....	336
Table 3-359. Function exmc_sqipssram_write_cmd_send.....	337
Table 3-360. Function exmc_sqipssram_low_id_get.....	337
Table 3-361. Function exmc_sqipssram_low_id_get.....	338
Table 3-362. Function exmc_sqipssram_send_command_state_get.....	338
Table 3-363. Function exmc_interrupt_enable	339
Table 3-364. Function exmc_interrupt_disable	340
Table 3-365. Function exmc_flag_get	341
Table 3-366. Function exmc_flag_clear	342
Table 3-367. Function exmc_interrupt_flag_get	343
Table 3-368. Function exmc_interrupt_flag_clear	344
Table 3-369. EXTI Registers.....	345
Table 3-370. EXTI firmware function	345
Table 3-371. Function exti_deinit.....	347
Table 3-372. Function exti_init.....	347
Table 3-373. Function exti_interrupt_enable	348
Table 3-374. Function exti_interrupt_disable	348
Table 3-375. Function exti_event_enable.....	349
Table 3-376. Function exti_event_disable.....	349
Table 3-377. Function exti_software_interrupt_enable	350
Table 3-378. Function exti_software_interrupt_disable	350
Table 3-379. Function exti_flag_get	351
Table 3-380. Function exti_flag_clear	351
Table 3-381. Function exti_interrupt_flag_get.....	352
Table 3-382. Function exti_interrupt_flag_clear	352
Table 3-383. FMC Registers.....	353
Table 3-384. FMC firmware function	353
Table 3-385. fmc_state_enum.....	354
Table 3-386. Function fmc_wscnt_set	355
Table 3-387. Function fmc_unlock.....	355
Table 3-388. Function fmc_lock	356
Table 3-389. Function fmc_page_erase	356
Table 3-390. Function fmc_sector_erase	357

Table 3-391. Function fmc_mass_erase	357
Table 3-392. Function fmc_bank0_erase	358
Table 3-393. Function fmc_bank1_erase	358
Table 3-394. Function fmc_word_program	359
Table 3-395. Function fmc_halfword_program	359
Table 3-396. Function fmc_halfword_program	360
Table 3-397. Function ob_unlock	361
Table 3-398. Function ob_lock	361
Table 3-399. Function ob_start	362
Table 3-400. Function ob_erase	362
Table 3-401. Function ob_write_protection_enable	363
Table 3-402. Function ob_write_protection_disable	363
Table 3-403. Function ob_drp_enable	364
Table 3-404. Function ob_drp_disable	365
Table 3-405. Function ob_security_protection_config	365
Table 3-406. Function ob_user_write	366
Table 3-407. Function ob_user_bor_threshold	367
Table 3-408. Function ob_boot_mode_config	368
Table 3-409. Function ob_user_get	368
Table 3-410. Function ob_write_protection0_get	369
Table 3-411. Function ob_write_protection1_get	369
Table 3-412. Function ob_drp0_get	370
Table 3-413. Function ob_drp0_get	370
Table 3-414. Function ob_spc_get	371
Table 3-415. Function ob_user_bor_threshold_get	372
Table 3-416. Function fmc_flag_get	372
Table 3-417. Function fmc_flag_clear	373
Table 3-418. Function fmc_interrupt_enable	373
Table 3-419. Function fmc_interrupt_disable	374
Table 3-420. Function fmc_interrupt_flag_get	375
Table 3-421. Function fmc_interrupt_flag_clear	375
Table 3-422. Function fmc_state_get	376
Table 3-423. Function fmc_ready_wait	377
Table 3-425. FWDGT Registers	377
Table 3-426. FWDGT firmware function	378
Table 3-427. Function fwdgt_write_ensable	378
Table 3-428. Function fwdgt_write_disable	379
Table 3-429. Function fwdgt_enable	379
Table 3-430. Function fwdgt_prescaler_value_config	380
Table 3-431. Function fwdgt_reload_value_config	381
Table 3-432. Function fwdgt_counter_reload	381
Table 3-433. Function fwdgt_config	382
Table 3-434. Function fwdgt_flag_get fwdgt_write_disable	383
Table 3-432. GPIO Registers	384

Table 3-433. GPIO firmware function	384
Table 3-434. Function gpio_deinit	385
Table 3-435. Function gpio_mode_set.....	385
Table 3-436. Function gpio_output_options_set	386
Table 3-437. Function gpio_bit_set	387
Table 3-438. Function gpio_bit_reset	388
Table 3-439. Function gpio_bit_write.....	388
Table 3-440. Function gpio_port_write	389
Table 3-441. Function gpio_input_bit_get.....	389
Table 3-442. Function gpio_input_port_get.....	390
Table 3-443. Function gpio_output_bit_get.....	391
Table 3-444. Function gpio_output_port_get	391
Table 3-445. Function gpio_pin_remap_config.....	392
Table 3-446. Function gpio_pin_lock	393
Table 3-447. Function gpio_bit_toggle	393
Table 3-448. Function gpio_port_toggle	394
Table 3-449. I2C Registers	395
Table 3-450. I2C firmware function.....	395
Table 3-451. Function i2c_deinit	397
Table 3-452. Function i2c_clock_config.....	398
Table 3-453. Function i2c_mode_addr_config	399
Table 3-454. Function i2c_smbus_type_config	400
Table 3-455. Function i2c_ack_config	400
Table 3-456. Function i2c_ackpos_config	401
Table 3-457. Function i2c_master_addressing	402
Table 3-458. Function i2c_dualaddr_enable	403
Table 3-459. Function i2c_enable	404
Table 3-460. Function i2c_disable	404
Table 3-461. Function i2c_start_on_bus	405
Table 3-462. Function i2c_stop_on_bus	406
Table 3-463. Function i2c_data_transmit	406
Table 3-464. Function i2c_data_receive	407
Table 3-465. Function i2c_dma_config.....	408
Table 3-466. Function i2c_dma_last_transfer_enable	409
Table 3-467. Function i2c_stretch_scl_low_config	409
Table 3-468. Function i2c_slave_response_to_gcall_config.....	410
Table 3-469. Function i2c_software_reset_config.....	411
Table 3-470. Function i2c_pec_config	412
Table 3-471. Function i2c_pec_transfer_config.....	413
Table 3-472. Function i2c_pec_value_get	413
Table 3-473. Function i2c_smbus_alert_config	414
Table 3-474. Function i2c_smbus_arp_config	415
Table 3-475. Function i2c_analog_noise_filter_disable	416
Table 3-476. Function i2c_analog_noise_filter_enable	416

Table 3-477. Function i2c_digital_noise_filter_config	417
Table 3-478. Function i2c_sam_enable	418
Table 3-479. Function i2c_sam_disable	418
Table 3-480. Function i2c_sam_timeout_enable.....	419
Table 3-481. Function i2c_sam_timeout_disable	420
Table 3-482. Function i2c_flag_get	420
Table 3-483. Function i2c_flag_clear	422
Table 3-484. Function i2c_interrupt_enable	423
Table 3-485. Function i2c_interrupt_disable	424
Table 3-486. Function i2c_interrupt_flag_get.....	425
Table 3-487. Function i2c_interrupt_flag_clear.....	427
Table 3-488. IPA Registers	428
Table 3-489. IPA firmware function.....	429
Table3-490. ipa_foreground_parameter_struct.....	430
Table3-491. ipa_background_parameter_struct	431
Table3-492. ipa_destination_parameter_struct.....	431
Table 3-493. Function ipa_deinit.....	432
Table 3-494. Function ipa_transfer_enable.....	432
Table 3-495. Function ipa_transfer_hangup_enable	433
Table 3-496. Function ipa_transfer_hangup_disable	434
Table 3-497. Function ipa_transfer_stop_enable.....	434
Table 3-498. Function ipa_transfer_stop_disable	435
Table 3-499. Function ipa_foreground_lut_loading_enable	435
Table 3-500. Function ipa_background_lut_loading_enable	436
Table 3-501. Function ipa_pixel_format_convert_mode_set.....	437
Table 3-502. Function ipa_foreground_struct_para_init	437
Table 3-503. Function ipa_foreground_init.....	438
Table 3-504. Function ipa_background_struct_para_init.....	439
Table 3-505. Function ipa_background_init	440
Table 3-506. Function ipa_destination_struct_para_init	441
Table 3-507. Function ipa_destination_init.....	442
Table 3-508. Function ipa_foreground_lut_init	443
Table 3-509. Function ipa_background_lut_init.....	444
Table 3-510. Function ipa_line_mark_config.....	445
Table 3-511. Function ipa_inter_timer_config	445
Table 3-512. Function ipa_interval_clock_num_config	446
Table 3-513. Function ipa_flag_get	447
Table 3-514. Function ipa_flag_clear	447
Table 3-515. Function ipa_interrupt_enable	448
Table 3-516. Function ipa_interrupt_disable	449
Table 3-517. Function ipa_interrupt_flag_get.....	450
Table 3-518. Function ipa_interrupt_flag_clear	451
Table 3-519. IREF Registers	452
Table 3-520. IREF firmware function	452

Table 3-521. Function iref_deinit	452
Table 3-522. Function iref_enable	453
Table 3-523. Function iref_disable	453
Table 3-524. Function iref_mode_set.....	454
Table 3-525. Function iref_sink_set	455
Table 3-526. Function iref_precision_trim_value_set.....	455
Table 3-527. Function iref_step_data_config	456
Table 3-528. NVIC Registers	456
Table 3-529. SysTick Registers.....	457
Table 3-530. Enum IRQn_Type	457
Table 3-531. MISC firmware function	460
Table 3-532. Function nvic_priority_group_set	460
Table 3-533. Function nvic_irq_enable.....	461
Table 3-534. Function nvic_irq_disable.....	461
Table 3-535. Function nvic_vector_table_set.....	462
Table 3-536. Function system_lowpower_set	462
Table 3-537. Function system_lowpower_reset.....	463
Table 3-538. Function systick_clksource_set	464
Table 3-539. PMU Registers.....	464
Table 3-540. PMU firmware function	465
Table 3-541. Function pmu_deinit	466
Table 3-542. Function pmu_lvd_select.....	466
Table 3-543. Function pmu_ldo_output_select.....	467
Table 3-544. Function pmu_low_driver_mode_enable	468
Table 3-545. Function pmu_highdriver_switch_select.....	469
Table 3-546. Function pmu_highdriver_mode_enable	469
Table 3-547. Function pmu_highdriver_mode_disable	470
Table 3-548. Function pmu_lvd_disable.....	470
Table 3-549. Function pmu_lowdriver_lowpower_config	471
Table 3-550. pmu_lowdriver_normalpower_config	472
Table 3-551. Function pmu_to_sleepmode.....	473
Table 3-552. Function pmu_to_deepsleepmode	473
Table 3-553. Function pmu_to_standbymode	474
Table 3-554. Function pmu_backup_ldo_config	475
Table 3-555. Function pmu_flag_reset	475
Table 3-556. Function pmu_flag_get.....	476
Table 3-557. Function pmu_backup_write_enable	477
Table 3-558. Function pmu_backup_write_disable	478
Table 3-559. Function pmu_wakeup_pin_enable	478
Table 3-560. Function pmu_wakeup_pin_disable	479
Table 3-561. RCU Registers	480
Table 3-562. RCU firmware function	481
Table 3-563. Enum rcu_periph_enum	483
Table 3-565. Enum rcu_periph_sleep_enum	484

Table 3-566. Enum rcu_periph_reset_enum	485
Table 3-567. Enum rcu_flag_enum	485
Table 3-568. Enum rcu_int_flag_enum	486
Table 3-569. Enum rcu_int_flag_clear_enum	487
Table 3-570. Enum rcu_int_enum	487
Table 3-571. Enum rcu_osc1_type_enum	487
Table 3-572. Enum rcu_clock_freq_enum	488
Table 3-573. Function rcu_deinit	488
Table 3-574. Function rcu_periph_clock_enable	489
Table 3-575. Function rcu_periph_clock_disable	489
Table 3-576. Function rcu_periph_clock_sleep_enable	490
Table 3-577. Function rcu_periph_clock_sleep_disable	490
Table 3-578. Function rcu_periph_reset_enable	491
Table 3-579. Function rcu_periph_reset_disable	492
Table 3-580. Function rcu_bkp_reset_enable	492
Table 3-581. Function rcu_bkp_reset_disable	493
Table 3-582. Function rcu_system_clock_source_config	494
Table 3-583. Function rcu_system_clock_source_get	494
Table 3-584. Function rcu_ahb_clock_config	495
Table 3-585. Function rcu_apb1_clock_config	496
Table 3-586. Function rcu_apb2_clock_config	497
Table 3-587. Function rcu_ckout0_config	498
Table 3-588. Function rcu_ckout1_config	499
Table 3-589. Function rcu_pll_config	500
Table 3-590. Function rcu_pll2s_config	501
Table 3-591. Function rcu_pll2sai_config	501
Table 3-592. Function rcu_rtc_clock_config	502
Table 3-593. Function rcu_i2s_clock_config	503
Table 3-594. Function rcu_ck48m_clock_config	504
Table 3-595. Function rcu_pll48m_clock_config	505
Table 3-596. Function rcu_timer_clock_prescaler_config	505
Table 3-597. Function rcu_tli_clock_div_config	506
Table 3-598. Function rcu_flag_get	507
Table 3-599. Function rcu_all_reset_flag_clear	507
Table 3-600. Function rcu_interrupt_flag_get	508
Table 3-601. Function rcu_interrupt_flag_clear	509
Table 3-602. Function rcu_interrupt_enable	509
Table 3-603. Function rcu_interrupt_disable	510
Table 3-604. Function rcu_lxtal_drive_capability_config	511
Table 3-605. Function rcu_osc1_stab_wait	511
Table 3-606. Function rcu_osc1_on	512
Table 3-607. Function rcu_osc1_off	513
Table 3-608. Function rcu_osc1_bypass_mode_enable	513
Table 3-609. Function rcu_osc1_bypass_mode_disable	514

Table 3-610. Function rcu_hxtal_clock_monitor_enable	515
Table 3-611. Function rcu_hxtal_clock_monitor_disable.....	515
Table 3-612. Function rcu_irc16m_adjust_value_set.....	516
Table 3-613. Function rcu_spread_spectrum_config	516
Table 3-614. Function rcu_spread_spectrum_enable.....	517
Table 3-615. Function rcu_spread_spectrum_disable.....	518
Table 3-616. Function rcu_voltage_key_unlock	519
Table 3-617. Function rcu_deepsleep_voltage_set.....	519
Table 3-618. Function rcu_clock_freq_get.....	520
Table 3-619. RTC Registers	521
Table 3-620.RTC firmware function.....	522
Table 3-621. rtc_parameter_struct.....	524
Table 3-622. rtc_alarm_struct.....	524
Table 3-623. rtc_timestamp_struct.....	524
Table 3-624. rtc_tamper_struct	525
Table 3-625. Function rtc_deinit	525
Table 3-626. Function rtc_init.....	526
Table 3-627. Function rtc_init_mode_enter	527
Table 3-628. Function rtc_init_mode_exit	527
Table 3-629. Function rtc_register_sync_wait	528
Table 3-630. Function rtc_current_time_get.....	529
Table 3-631. Function rtc_subsecond_get.....	529
Table 3-632. Function rtc_alarm_config	530
Table 3-633. Function rtc_alarm_subsecond_config	531
Table 3-634. Function rtc_alarm_get.....	532
Table 3-635. Function rtc_alarm_subsecond_get	533
Table 3-636. Function rtc_alarm_enable	534
Table 3-637. Function rtc_alarm_disable	534
Table 3-638. Function rtc_timestamp_enable	535
Table 3-639. Function rtc_timestamp_disable	536
Table 3-640. Function rtc_timestamp_get.....	536
Table 3-641. Function rtc_timestamp_subsecond_get.....	537
Table 3-642. Function rtc_timestamp_pin_map.....	538
Table 3-643. Function rtc_timestamp_enable	539
Table 3-644. Function rtc_tamper_disable.....	539
Table 3-645. Function rtc_tamper0_pin_map	540
Table 3-646. Function rtc_interrupt_enable.....	541
Table 3-647. Function rtc_interrupt_disable	541
Table 3-648. Function rtc_flag_get	542
Table 3-649. Function rtc_flag_clear	543
Table 3-650. Function rtc_alter_output_config	544
Table 3-651. rtc_calibration_output_config	545
Table 3-652. rtc_hour_adjust.....	546
Table 3-653. rtc_second_adjust.....	547

Table 3-654. rtc_bypass_shadow_enable	547
Table 3-655. rtc_bypass_shadow_disable	548
Table 3-656. rtc_refclock_detection_enable	549
Table 3-657. rtc_refclock_detection_disable	549
Table 3-658. rtc_wakeup_enable	550
Table 3-659. rtc_wakeup_disable	550
Table 3-660. rtc_wakeup_clock_set	551
Table 3-661. rtc_wakeup_timer_set	552
Table 3-662. rtc_wakeup_timer_get	553
Table 3-663. rtc_smooth_calibration_config	553
Table 3-664. rtc_coarse_calibration_enable	554
Table 3-665. rtc_coarse_calibration_disable	555
Table 3-666. rtc_coarse_calibration_config	555
Table 3-667. SDIO Registers	557
Table 3-668. SDIO firmware function	557
Table 3-669. Function sdio_deinit	559
Table 3-670. Function sdio_clock_config	560
Table 3-671. Function sdio_hardware_clock_enable	561
Table 3-672. Function sdio_hardware_clock_disable	562
Table 3-673. Function sdio_bus_mode_set	562
Table 3-674. Function sdio_power_state_set	563
Table 3-675. Function sdio_power_state_get	564
Table 3-676. Function sdio_clock_enable	565
Table 3-677. Function sdio_clock_disable	565
Table 3-678. Function sdio_command_response_config	566
Table 3-679. Function sdio_wait_type_set	567
Table 3-680. Function sdio_csm_enable	568
Table 3-681. Function sdio_csm_disable	568
Table 3-682. Function sdio_command_index_get	569
Table 3-683. Function sdio_response_get	569
Table 3-684. Function sdio_data_config	570
Table 3-685. Function sdio_data_transfer_config	572
Table 3-686. Function sdio_dsm_enable	573
Table 3-687. Function sdio_dsm_disable	573
Table 3-688. Function sdio_data_write	574
Table 3-689. Function sdio_data_read	575
Table 3-690. Function sdio_data_counter_get	575
Table 3-691. Function sdio_data_counter_get	576
Table 3-692. Function sdio_dma_enable	577
Table 3-693. Function sdio_dma_disable	577
Table 3-694. Function sdio_flag_get	578
Table 3-695. Function sdio_flag_clear	579
Table 3-696. Function sdio_interrupt_enable	581
Table 3-697. Function sdio_interrupt_disable	582

Table 3-698. Function sdio_interrupt_flag_get	584
Table 3-699. Function sdio_interrupt_flag_clear	586
Table 3-700. Function sdio_readwait_enable	587
Table 3-701. Function sdio_readwait_disable	588
Table 3-702. Function sdio_stop_readwait_enable	588
Table 3-703. Function sdio_stop_readwait_disable	589
Table 3-704. Function sdio_readwait_type_set.....	589
Table 3-705. Function sdio_operation_enable	590
Table 3-706. Function sdio_operation_disable	591
Table 3-707. Function sdio_suspend_enable	591
Table 3-708. Function sdio_suspend_disable.....	592
Table 3-709. Function sdio_ceata_command_enable.....	593
Table 3-710. Function sdio_ceata_command_disable.....	593
Table 3-711. Function sdio_ceata_interrupt_enable	594
Table 3-712. Function sdio_ceata_interrupt_disable	594
Table 3-713. Function sdio_ceata_command_completion_enable	595
Table 3-714. Function sdio_ceata_command_completion_disable	596
Table 3-715. SPI/I2S Registers	596
Table 3-716. SPI/I2S firmware function.....	597
Table 3-717. spi_parameter_struct.....	598
Table 3-718. Function spi_i2s_deinit	599
Table 3-719. Function spi_struct_para_init	599
Table 3-720. Function spi_init	600
Table 3-721. Function spi_enable	600
Table 3-722. Function spi_disable	601
Table 3-723. Function i2s_init	601
Table 3-724. Function i2s_psc_config	602
Table 3-725. Function i2s_enable	604
Table 3-726. Function i2s_disable	604
Table 3-727. Function spi_nss_output_enable	605
Table 3-728. Function spi_nss_output_disable	605
Table 3-729. Function spi_nss_internal_high	606
Table 3-730. Function spi_nss_internal_low	606
Table 3-731. Function spi_dma_enable	607
Table 3-732. Function spi_dma_disable	608
Table 3-733. Function spi_i2s_data_frame_format_config	608
Table 3-734. Function spi_i2s_data_transmit.....	609
Table 3-735. Function spi_i2s_data_receive	609
Table 3-736. Function spi_bidirectional_transfer_config.....	610
Table 3-737. Function spi_crc_polynomial_set.....	612
Table 3-738. Function spi_crc_polynomial_get	613
Table 3-739. Function spi_crc_on.....	614
Table 3-740. Function spi_crc_off	614
Table 3-741. Function spi_crc_next	615

Table 3-742. Function spi_crc_get.....	615
Table 3-743. Function spi_ti_mode_enable	616
Table 3-744. Function spi_ti_mode_disable	617
Table 3-745. Function i2s_full_duplex_mode_config	611
Table 3-746. Function qspi_enable	617
Table 3-747. Function qspi_disable	618
Table 3-748. Function qspi_write_enable.....	618
Table 3-749. Function qspi_read_enable	619
Table 3-750. Function qspi_io23_output_enable	619
Table 3-751. Function qspi_io23_output_disable	620
Table 3-752. Function spi_i2s_interrupt_enable.....	621
Table 3-753. Function spi_i2s_interrupt_disable.....	622
Table 3-754. Function spi_i2s_interrupt_flag_get	623
Table 3-755. Function spi_i2s_flag_get	620
Table 3-756. Function spi_crc_error_clear	616
Table 3-757. SYSCFG registers	624
Table 3-758. SYSCFG firmware function	624
Table 3-759. Function syscfg_deinit	625
Table 3-760. Function syscfg_bootmode_config.....	625
Table 3-761. Function syscfg_fmc_swap_config	626
Table 3-762. Function syscfg_exmc_swap_config.....	627
Table 3-763. Function syscfg_exti_line_config.....	628
Table 3-764. Function syscfg_enet_phy_interface_config	628
Table 3-765. Function syscfg_compensation_config	629
Table 3-766. Function syscfg_flag_get.....	630
Table 3-767. TIMERx Registers	631
Table 3-768. TIMERx firmware function.....	632
Table 3-769. Structure timer_parameter_struct	635
Table 3-770. Structure timer_break_parameter_struct.....	635
Table 3-771. Structure timer_oc_parameter_struct.....	636
Table 3-772. Structure timer_ic_parameter_struct.....	636
Table 3-773. Function timer_deinit	637
Table 3-774. Function timer_struct_para_init.....	637
Table 3-775. Function timer_init	638
Table 3-776. Function timer_enable	639
Table 3-777. Function timer_disable	639
Table 3-778. Function timer_auto_reload_shadow_enable	640
Table 3-779. Function timer_auto_reload_shadow_disable	641
Table 3-780. Function timer_update_event_enable	641
Table 3-781. Function timer_update_event_disable	642
Table 3-782. Function timer_counter_alignment	643
Table 3-783. Function timer_counter_up_direction	644
Table 3-784. Function timer_counter_down_direction	644
Table 3-785. Function timer_prescaler_config.....	645

Table 3-786. Function timer_repetition_value_config	646
Table 3-787. Function timer_autoreload_value_config	647
Table 3-788. Function timer_counter_value_config	647
Table 3-789. Function timer_counter_read	648
Table 3-790. Function timer_prescaler_read	649
Table 3-791. Function timer_single_pulse_mode_config	650
Table 3-792. Function timer_update_source_config.....	650
Table 3-793. Function timer_dma_enable	651
Table 3-794. Function timer_dma_disable	652
Table 3-795. Function timer_channel_dma_request_source_select.....	653
Table 3-796. Function timer_dma_transfer_config.....	654
Table 3-797. Function timer_event_software_generate.....	656
Table 3-798. Function timer_break_struct_para_init	657
Table 3-799. Function timer_break_config	658
Table 3-800. Function timer_break_enable	659
Table 3-801. Function timer_break_disable.....	660
Table 3-802. Function timer_automatic_output_enable	661
Table 3-803. Function timer_automatic_output_disable	661
Table 3-804. Function timer_primary_output_config	662
Table 3-805. Function timer_channel_control_shadow_config	663
Table 3-806. Function timer_channel_control_shadow_update_config.....	664
Table 3-807. Function timer_channel_output_struct_para_init	664
Table 3-808. Function timer_channel_output_config	665
Table 3-809. Function timer_channel_output_mode_config	666
Table 3-810. Function timer_channel_output_pulse_value_config.....	668
Table 3-811. Function timer_channel_output_shadow_config	669
Table 3-812. Function timer_channel_output_fast_config.....	670
Table 3-813. Function timer_channel_output_clear_config	671
Table 3-814. Function timer_channel_output_polarity_config.....	672
Table 3-815. Function timer_channel_complementary_output_polarity_config	673
Table 3-816. Function timer_channel_output_state_config	674
Table 3-817. Function timer_channel_complementary_output_state_config	675
Table 3-818. Function timer_channel_input_struct_para_init.....	676
Table 3-819. Function timer_input_capture_config.....	677
Table 3-820. Function timer_channel_input_capture_prescaler_config	678
Table 3-821. Function timer_channel_capture_value_register_read	679
Table 3-822. Function timer_input_pwm_capture_config.....	680
Table 3-823. Function timer_hall_mode_config.....	681
Table 3-824. Function timer_input_trigger_source_select	682
Table 3-825. Function timer_master_output_trigger_source_select	683
Table 3-826. Function timer_slave_mode_select	684
Table 3-827. Function timer_master_slave_mode_config	686
Table 3-828. Function timer_external_trigger_config	687
Table 3-829. Function timer_quadrature_decoder_mode_config.....	688

Table 3-830. Function timer_internal_clock_config	689
Table 3-831. Function timer_internal_trigger_as_external_clock_config	690
Table 3-832. Function timer_external_trigger_as_external_clock_config	691
Table 3-833. Function timer_external_clock_mode0_config	692
Table 3-834. Function timer_external_clock_mode1_config	693
Table 3-835. Function timer_external_clock_mode1_disable	694
Table 3-836. Function timer_channel_remap_config	695
Table 3-837. Function timer_write_chxval_register_config	696
Table 3-838. Function timer_output_value_selection_config	697
Table 3-839. Function timer_flag_get	698
Table 3-840. Function timer_flag_clear	699
Table 3-841. Function timer_interrupt_enable	700
Table 3-842. Function timer_interrupt_disable	701
Table 3-843. Function timer_interrupt_flag_get	702
Table 3-844. Function timer_interrupt_flag_clear	703
Table 3-845. TLI Registers	705
Table 3-846. TLI firmware function	706
Table 3-847. tli_parameter_struct	707
Table 3-848. tli_layer_parameter_struct	708
Table 3-849. tli_layer_lut_parameter_struct	708
Table 3-850. Function tli_deinit	709
Table 3-851. Function tli_struct_para_init	709
Table 3-852. Function tli_init	710
Table 3-853. Function tli_dither_config	711
Table 3-854. Function tli_enable	712
Table 3-855. Function tli_disable	712
Table 3-856. Function tli_reload_config	713
Table 3-857. Function tli_layer_struct_para_init	714
Table 3-858. Function tli_layer_init	714
Table 3-859. Function tli_layer_window_offset_modify	716
Table 3-860. Function tli_lut_struct_para_init	717
Table 3-861. Function tli_lut_init	717
Table 3-862. Function tli_color_key_init	718
Table 3-863. Function tli_layer_enable	719
Table 3-864. Function tli_layer_disable	720
Table 3-865. Function tli_color_key_enable	721
Table 3-866. Function tli_color_key_disable	721
Table 3-867. Function tli_lut_enable	722
Table 3-868. Function tli_lut_disable	722
Table 3-869. Function tli_line_mark_set	723
Table 3-870. Function tli_current_pos_get	724
Table 3-871. Function tli_interrupt_enable	724
Table 3-872. Function tli_interrupt_disable	725
Table 3-873. Function tli_interrupt_flag_get	726

Table 3-874. Function tli_interrupt_flag_clear.....	727
Table 3-875. Function tli_flag_get	728
Table 3-876 TRNG Registers	729
Table 3-877. TRNG firmware function.....	729
Table 3-878. Function trng_deinit.....	729
Table 3-879. Function trng_enable	730
Table 3-880 Function trng_disable	731
Table 3-881 Function trng_get_true_random_data	731
Table 3-882 Function trng_interrupt_enable	732
Table 3-883 Function trng_interrupt_disable	732
Table 3-884 Function trng_flag_get.....	733
Table 3-885 Function trng_interrupt_flag_get.....	734
Table 3-886 Function trng_interrupt_flag_clear.....	735
Table 3-887. USART Registers	736
Table 3-888. USART firmware function.....	736
Table 3-889. Function usart_deinit	738
Table 3-890. Function usart_baudrate_set	738
Table 3-891. Function usart_parity_config	739
Table 3-892. Function usart_word_length_set.....	739
Table 3-893. Function usart_stop_bit_set.....	740
Table 3-894. Function usart_enable	741
Table 3-895. Function usart_disable	741
Table 3-896. Function usart_transmit_config.....	742
Table 3-897. Function usart_receive_config	742
Table 3-898. Function usart_data_first_config	743
Table 3-899. Function usart_invert_config	744
Table 3-900. Function usart_oversample_config	744
Table 3-901. Function usart_sample_bit_config.....	745
Table 3-902. Function usart_receiver_timeout_enable.....	746
Table 3-903. Function usart_receiver_timeout_disable.....	746
Table 3-904. Function usart_receiver_timeout_threshold_config	747
Table 3-905. Function usart_data_transmit	747
Table 3-906. Function usart_data_receive	748
Table 3-907. Function usart_address_config	748
Table 3-908. Function usart_mute_mode_enable.....	749
Table 3-909. Function usart_mute_mode_disable.....	750
Table 3-910. Function usart_mute_mode_wakeup_config	750
Table 3-911. Function usart_lin_mode_enable	751
Table 3-912. Function usart_lin_mode_disable	751
Table 3-913. Function usart_lin_break_dection_length_config	752
Table 3-914. Function usart_send_break	753
Table 3-915. Function usart_halfduplex_enable	753
Table 3-916. Function usart_halfduplex_disable	754
Table 3-917. Function usart_synchronous_clock_enable	754

Table 3-918. Function usart_synchronous_clock_disable	755
Table 3-919. Function usart_synchronous_clock_config	755
Table 3-920. Function usart_guard_time_config	756
Table 3-921. Function usart_smartcard_mode_enable	757
Table 3-922. Function usart_smartcard_mode_disable	757
Table 3-923. Function usart_smartcard_mode_nack_enable.....	758
Table 3-924. Function usart_smartcard_mode_nack_disable.....	758
Table 3-925. Function usart_smartcard_autoretry_config.....	759
Table 3-926. Function usart_block_length_config	759
Table 3-927. Function usart_irda_mode_enable.....	760
Table 3-928. Function usart_irda_mode_disable.....	760
Table 3-929. Function usart_prescaler_config.....	761
Table 3-930. Function usart_irda_lowpower_config	761
Table 3-931. Function usart_hardware_flow_rts_config	762
Table 3-932. Function usart_hardware_flow_cts_config	763
Table 3-933. Function usart_break_frame_coherence_config	763
Table 3-934. Function usart_parity_check_coherence_config	764
Table 3-935. Function usart_hardware_flow_coherence_config	765
Table 3-936. Function usart_dma_receive_config.....	765
Table 3-937. Function usart_dma_transmit_config.....	766
Table 3-938. Function usart_flag_get	767
Table 3-939. Function usart_flag_clear	768
Table 3-940. Function usart_interrupt_enable	768
Table 3-941. Function usart_interrupt_disable	769
Table 3-942. Function usart_interrupt_flag_get.....	770
Table 3-943. Function usart_interrupt_flag_clear.....	771
Table 3-944. WWDGT Registers	772
Table 3-945. WWDGT firmware function	772
Table 3-946. Function wwdgt_deinit	773
Table 3-947. Function wwdgt_enable	774
Table 3-948. Function wwdgt_counter_update	774
Table 3-949. Function wwdgt_config	775
Table 3-950. Function wwdgt_interrupt_enable	776
Table 3-951. Function wwdgt_flag_get	776
Table 3-952. Function wwdgt_flag_clear	777
Table 4-1. Revision history	779

1. Introduction

This manual introduces firmware library of GD32F4xx devices which are 32-bit microcontrollers based on the ARM processor.

The firmware library is a firmware function package, including program, data structure and macro definitions, all the performance features of peripherals of GD32F4xx devices are involved in the package. The peripheral driving code and firmware examples on evaluation board are also included in firmware library. Users need not learn each peripherals in details and it's easy to apply a peripheral by using the firmware library. Using firmware library can greatly reduce programming time, thereby reducing development costs.

The driving code of each peripheral is concluded by a group of functions, which describes all the performance features of the peripheral. Users can drive a peripheral by a group of APIs (application programming interface), all the APIs are standardized about the code structure, function name and parameter names.

All the driving source code accord with MISRA-C:2004 standard (example files accord with extended ANSI-C standard), and will not be influenced by differences of IDEs, except the startup files which are written differently according to the IDEs.

The commonly used firmware library includes all the functions of all the peripherals, so the code size and the execution speed may not be the optimal. For most applications, users can use the library functions directly, while for the applications which are strict with the code size and execution speed, the firmware library can be used as the reference resource of how to configure a peripheral, and users adjust the code according to actual needs.

The overall structure of the firmware library user manual is shown as below:

- Rules of user manual and firmware library;
- Firmware library overview;
- Functions and registers descriptions of firmware library.

1.1. Rules of User Manual and Firmware Library

1.1.1. Peripherals

Table 1-1. Peripherals

Peripherals	Descriptions
ADC	Analog-to-digital converter
CAN	Controller area network
CRC	CRC calculation unit
CTC	Clock trim controller
DAC	Digital-to-analog converter

Peripherals	Descriptions
DBG	Debug
DCI	Digital camera interface
DMA	Direct memory access controller
ENET	Ethernet
EXMC	External memory controller
EXTI	Interrupt/event controller
FMC	Flash memory controller
FWDGT	Free watchdog timer
GPIO/APIO	General-purpose and alternate-function I/Os
I2C	Inter-integrated circuit interface
IPA	Image processing accelerator
IREF	Programmable current reference
MISC	Nested Vectored Interrupt Controller
PMU	Power management unit
RCU	Reset and clock unit
RTC	Real-time Clock
SDIO	Secure digital input/output interface
SPI/I2S	Serial peripheral interface/Inter-IC sound
SYSCFG	System configuration
TIMER	TIMER
TLI	TFT-LCD interface
TRNG	True random number generator
USART	Universal synchronous/asynchronous receiver /transmitter
WWDGT	Window watchdog timer
USBFS	Universal serial bus full-speed interface

1.1.2. Naming rules

The firmware library naming rules are shown as below:

- The peripherals are shortened in XXX format, such as: ADC. More shorten information of peripherals refer to [Peripherals](#);
- The name of sourcefile and header file are started with “gd32f4xx_”, such as: gd32f4xx_adc.h;
- The constants used only in one file should be defined in the used file; the constants used in many files should be defined in corresponding header file. All the constants are written in uppercase of English letters;
- Registers are handled as constants. The naming of them are written in uppercase of English letters. In most cases, register names are shortened accord with the user manual;
- Variables are written in lowercase, when concluded by several words, underlines should be adapted among words;
- The naming of peripheral functions are started with the peripheral abbreviation added

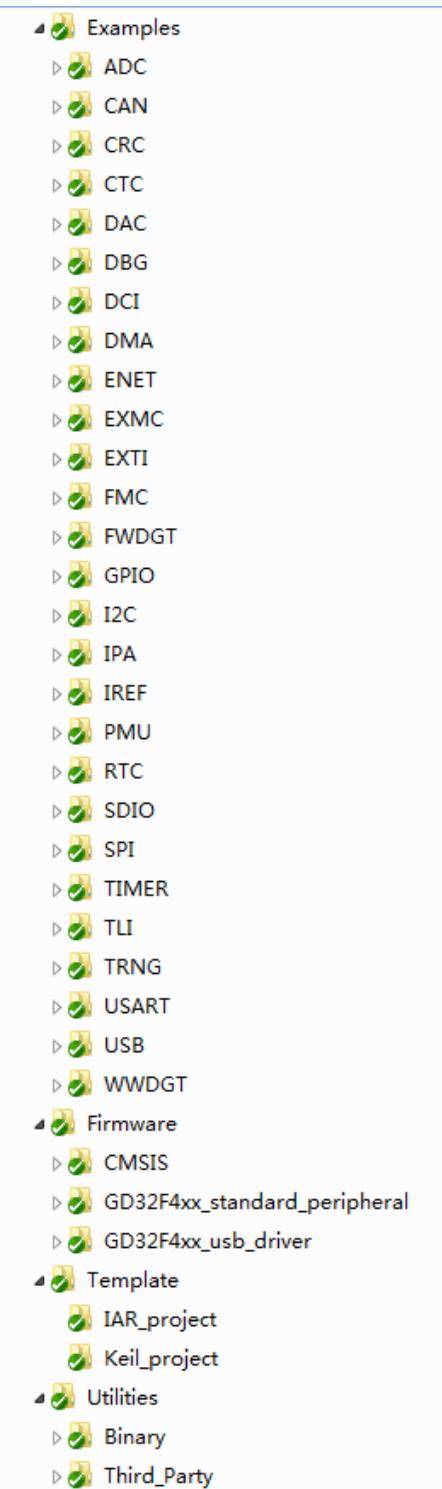
with an underline, when the function name is concluded by several words, underlines should be adapted among words, and all the peripheral functions are written in lowercase.

2. Firmware Library Overview

2.1. File Structure of Firmware Library

GD32F4xx_Firmware_Library, the file structure is shown as below:

Figure 2-1. File structure of firmware library of GD32F4xx



2.1.1. Examples Folder

Examples folder, each of GD32 peripheral has a subfolder. Each subfolder contains one or more examples of the peripheral, to show how to use the peripheral correctly. Each of the example subfolder includes the files shown as below:

- readme.txt: the description and using guide of the example;
- gd32f4xx_libopt.h: the header file configures all the peripherals used in the example, included by different “DEFINE” sentences (all the peripherals are enabled by default);
- gd32f4xx_it.c: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- gd32f4xx_it.h: the header file include all the prototypes of the interrupt service routines;
- systick.c: the source file include the precise time delay functions by using systick;
- systick.h: the header file include the prototype of the precise time delay functions by using systick;
- main.c: example code. Note: all the examples are not influenced by software IDEs.

2.1.2. Firmware Folder

Firmware folder includes all the subfolder and files which are the core part of the firmware:

- CMSIS subfolder includes the Cortex M4 kernel support files, the startup file based on the Cortex M4 kernel processor, the global header file of GD32F4xx and system configuration file;
- GD32F4xx_standard_peripheral subfolder:
 - Include subfolder includes all the header files of firmware library, users need not modify this folder;
 - Source subfolder includes all the source files of firmware library, users need not modify this folder;
- GD32F4xx_usbfs_driver subfolder includes all the related files about USBFS peripheral:
 - Include subfolder includes the header files of USBFS peripheral, users need not modify this folder;
 - Source subfolder includes the source files of USBFS peripheral, users need not modify this folder;

Note: All the codes accord with MISRA-C:2004 standard, and will not be influenced by different software IDEs.

2.1.3. Project Folder

Project folder includes examples of USBFS (EWARM is run in IAR, and MDK-ARM is run in Keil4).

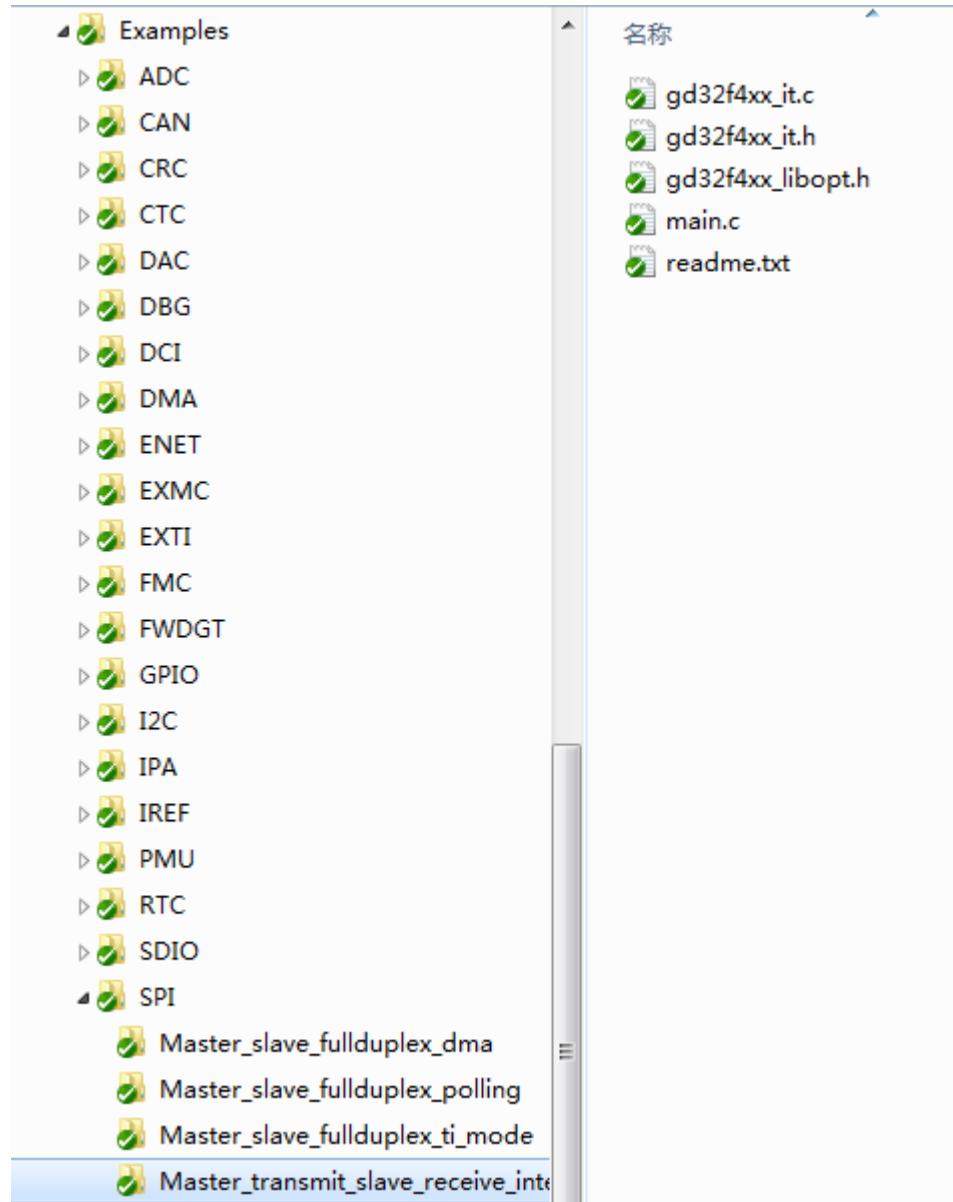
2.1.4. Template Folder

Template folder includes a simple demo of how to use LED, how to print by USART and use key to control, (IAR_project is run in IAR, and Keil_project is run in Keil4). User can use the project template to compile the firmware examples, the steps are shown as below:

Select files

Open “Examples” folder, select the module to be tested, such as SPI, open “SPI” folder, select an example of SPI, such as “SPI_master_transmit_slave_receive_interrupt”, shown as below:

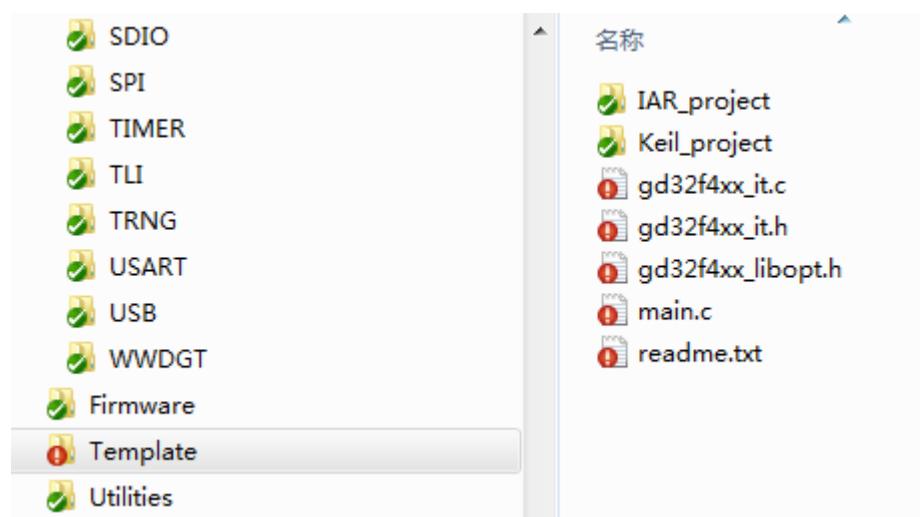
Figure 2-2. Select peripheral example files



Copy files

Open “Template” folder, keep the folders of “IAR_project” and “Keil_project”, and delete the other files, then copy all the files in “SPI_master_transmit_slave_receive_interrupt” folder to the “Template” subfolder, shown as below:

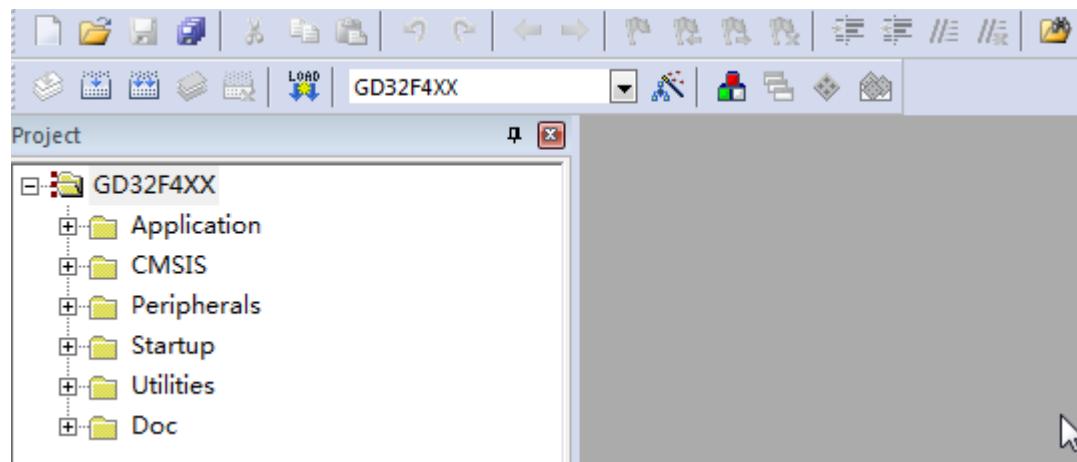
Figure 2-3. Copy the peripheral example files



Open a project

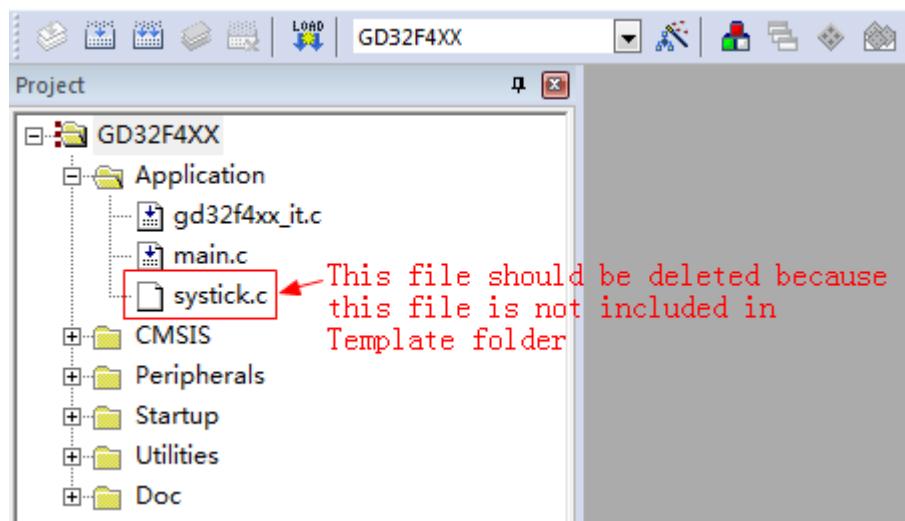
GD provides project in Keil and IAR, users can open project in different IDEs according to their need, such as "Keil_project", open \Template\Keil_project\Project.uvproj, shown as below:

Figure 2-4. Open the project file



Because different module and different functions adopt different files, users should add or delete the files in project according to the copied files, shown as below:

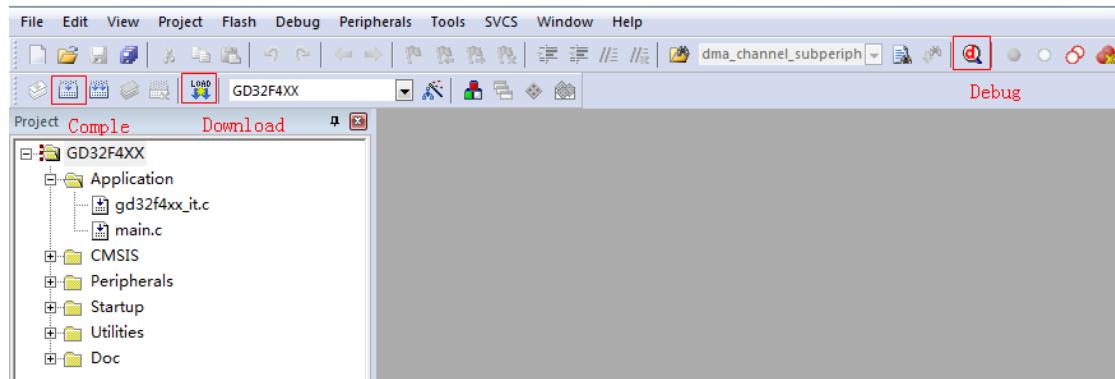
Figure 2-5. Configure project files



Compile-Debug-Download

First compile the project, if there is no error, then select the right jumper cap according to the description of readme, download the project to the target board, and there will be the phenomenon showed accord with the description of readme. The usage of IDE can refer to corresponding software user guide. If users are using Keil, the figure is shown as below:

Figure 2-6. Compile-debug-download



2.1.5. Utilities Folder

Utilities folder includes files about the firmware examples on evaluation board:

- Binary, Third_Party subfolders include files for USB tests;
- gd32f4xx_eval.h is related header files of the evaluation board about running the firmware examples;
- gd32f4xx_eval.c is related source files of the evaluation board about running the firmware examples.

Note: All the codes accord with MISRA-C:2004 standard, and will not be influenced by

different software IDEs.

2.2. File descriptions of Firmware Library

The major files about the firmware library are listed and described in the table below.

Table 2-1. Function descriptions of Firmware Library

Files	Descriptions
gd32f4xx_libopt.h	The header file about all the header files of peripherals. It is the only one file which is necessity to be included in the user's application, to connect the firmware library and the application.
main.c	Example of main function.
gd32f4xx_it.h	Header file, including all the prototypes of interrupt service routines.
gd32f4xx_it.c	Source files about interrupt service routines of peripherals. User can written his own interrupt functions in this file. For the different interrupt service requests to the same interrupt vector, users can confirm the interrupt source by functions of judging interrupt flags of peripherals. The functions are included in the firmware library.
gd32f4xx_xxx.h	The header file of peripheral PPP, including functions about peripheral PPP, and the variables used for functions.
gd32f4xx_xxx.c	The C source file for driving peripheral PPP.
systick.h	The header file of systick.c, including prototypes of systick configuration function and delay function.
systick.c	The source file about systick configuration function and delay function.
readme.txt	Description document about how to configure and how to use the firmware example.

3. Firmware Library of Standard Peripherals

3.1. Overview of Firmware Library of Standard Peripherals

The description format of firmware functions are shown as below:

Table 3-1. Peripheral function format of Firmware Library

Function name	Name of peripheral function
Function prototype	Declaration prototype
Function descriptions	Explain the function how to work
Precondition	Requirements should meet before calling this function
The called functions	Other firmware functions called in this function
Input parameter{in}	
Input parameter name	Description
xxxx	Description of input parameters
Output parameter{out}	
Output parameter name	Description
xxxx	Description of output parameters
Return value	
Return value type	The range of return value

3.2. ADC

The 12-bit ADC is an analog-to-digital converter using the successive approximation method. The ADC registers are listed in chapter [3.2.1](#) the ADC firmware functions are introduced in chapter [3.2.2](#).

3.2.1. Descriptions of Peripheral registers

ADC registers are listed in the table shown as below:

Table 3-2. ADC Registers

Registers	Descriptions
ADC_STAT	Status register
ADC_CTL0	Control register 0
ADC_CTL1	Control register 1
ADC_SAMPT0	Sample time register 0
ADC_SAMPT1	Sample time register 1
ADC_IOFFx (x=0..3)	Inserted channel data offset register x(x=0..3)
ADC_WDHT	Watchdog high threshold register

Registers	Descriptions
ADC_WDLT	Watchdog low threshold register
ADC_RSQ0	Regular sequence register 0
ADC_RSQ1	Regular sequence register 1
ADC_RSQ2	Regular sequence register 2
ADC_ISQ	Inserted sequence register
ADC_IDATAx	Inserted data register x(x=0..3)
ADC_RDATA	Regular data register
ADC_OVSAMPCTL	Oversample control register
ADC_SSTAT	Summary status register
ADC_SYNCCTL	Sync control register
ADC_SYNCDATA	Sync regular data register

3.2.2. Descriptions of Peripheral functions

ADC firmware functions are listed in the table shown as below:

Table 3-3. ADC firmware function

Function name	Function description
adc_deinit	reset ADCx peripheral
adc_clock_config	configure the ADC clock for all the ADCs
adc_special_function_config	enable or disable ADC special function
adc_data_alignment_config	configure ADC data alignment
adc_enable	enable ADC interface
adc_disable	disable ADC interface
adc_calibration_enable	ADC calibration and reset calibration
adc_channel_16_to_18	configure temperature sensor and internal reference voltage channel or VBAT channel function
adc_resolution_config	configure ADC resolution
adc_oversample_mode_config	configure ADC oversample mode
adc_oversample_mode_enable	enable ADC oversample mode
adc_oversample_mode_disable	disable ADC oversample mode
adc_dma_mode_enable	enable DMA request
adc_dma_mode_disable	disable DMA request
adc_dma_request_after_last_enable	when DMA=1, the DMA engine issues a request at end of each regular conversion
adc_dma_request_after_last_disable	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
adc_discontinuous_mode_config	configure ADC discontinuous mode
adc_channel_length_config	configure the length of regular channel group or inserted channel group
adc_regular_channel_config	configure ADC regular channel

Function name	Function description
adc_inserted_channel_config	configure ADC inserted channel
adc_inserted_channel_offset_config	configure ADC inserted channel offset
adc_external_trigger_source_config	configure ADC external trigger source
adc_external_trigger_config	enable ADC external trigger
adc_software_trigger_enable	enable ADC software trigger
adc_end_of_conversion_config	configure end of conversion mode
adc_regular_data_read	read ADC regular group data register
adc_inserted_data_read	read ADC inserted group data register
adc_watchdog_single_channel_disable	disable ADC analog watchdog single channel
adc_watchdog_single_channel_enable	configure ADC analog watchdog single channel
adc_watchdog_group_channel_enable	configure ADC analog watchdog group channel
adc_watchdog_disable	disable ADC analog watchdog
adc_watchdog_threshold_config	configure ADC analog watchdog threshold
adc_flag_get	get the ADC flag bits
adc_flag_clear	clear the ADC flag bits
adc_regular_software_startconv_flag_get	get the bit state of ADCx software start conversion
adc_inserted_software_startconv_flag_get	get the bit state of ADCx software inserted channel start conversion
adc_interrupt_flag_get	get the ADC interrupt bits
adc_interrupt_flag_clear	clear the ADC flag
adc_interrupt_enable	enable ADC interrupt
adc_interrupt_disable	disable ADC interrupt
adc_sync_mode_config	configure the ADC sync mode
adc_sync_delay_config	configure the delay between 2 sampling phases in ADC sync modes
adc_sync_dma_config	configure ADC sync DMA mode selection
adc_sync_dma_request_after_last_enable	when SYNC DMA is not equal to 2'b00, the DMA engine issues requests according to the SYNC DMA bits
adc_sync_dma_request_after_last_disable	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
adc_sync_regular_data_read	read ADC sync regular data register

adc_deinit

The description of adc_deinit is shown as below:

Table 3-4. Function adc_deinit

Function name	adc_deinit

Function prototype	void adc_deinit(void);
Function descriptions	reset ADC peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset ADC */
adc_deinit();
```

adc_clock_config

The description of `adc_clock_config` is shown as below:

Table 3-5. Function `adc_clock_config`

Function name	adc_clock_config
Function prototype	void adc_clock_config(uint32_t prescaler);
Function descriptions	configure the ADC clock for all the ADCs
Precondition	-
The called functions	-
Input parameter{in}	
prescaler	configure ADCs prescaler ratio
<i>ADC_ADCCK_PCLK2_DIV2</i>	PCLK2 div2
<i>ADC_ADCCK_PCLK2_DIV4</i>	PCLK2 div4
<i>ADC_ADCCK_PCLK2_DIV6</i>	PCLK2 div6
<i>ADC_ADCCK_PCLK2_DIV8</i>	PCLK2 div8
<i>ADC_ADCCK_HCLK_DIV5</i>	HCLK div5

<i>ADC_ADCCK_HCLK_D /V6</i>	HCLK div6
<i>ADC_ADCCK_HCLK_D /V10</i>	HCLK div10
<i>ADC_ADCCK_HCLK_D /V20</i>	HCLK div20
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the ADC clock: PCLK2 div2 */
adc_clock_config (ADC_ADCCK_PCLK2_DIV2);
```

adc_special_function_config

The description of `adc_special_function_config` is shown as below:

Table 3-6. Function `adc_special_function_config`

Function name	adc_special_function_config
Function prototype	void adc_special_function_config(uint32_t adc_periph, uint32_t function, ControlStatus newvalue);
Function descriptions	enable or disable ADC special function
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
function	the function to config
ADC_SCAN_MODE	scan mode select
ADC_INSERTED_CHANNEL_AUTO	inserted channel group convert automatically

<i>ADC_CONTINUOUS_MODE</i>	continuous mode select
Input parameter{in}	
<i>newvalue</i>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC0 scan mode */
adc_special_function_config(ADC0, ADC_SCAN_MODE, ENABLE);
```

adc_data_alignment_config

The description of `adc_data_alignment_config` is shown as below:

Table 3-7. Function `adc_data_alignment_config`

Function name	<code>adc_data_alignment_config</code>
Function prototype	<code>void adc_data_alignment_config(uint32_t adc_periph, uint32_t data_alignment);</code>
Function descriptions	configure ADCx data alignment
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
data_alignment	data alignment select
<i>ADC_DATAALIGN_RIGHT</i>	LSB alignment

<i>ADC_DATAALIGN_LT</i>	MSB alignment
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 data alignment */

adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);
```

adc_enable

The description of adc_enable is shown as below:

Table 3-8. Function adc_enable

Function name	adc_enable
Function prototype	void adc_enable(uint32_t adc_periph);
Function descriptions	enable ADCx interface
Precondition	-
The called functions	-
Input parameter{in}	
<i>adc_periph</i>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC0 */

adc_enable(ADC0);
```

adc_disable

The description of adc_disable is shown as below:

Table 3-9. Function adc_disable

Function name	adc_disable
Function prototype	void adc_disable(uint32_t adc_periph);
Function descriptions	disable ADCx interface
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 */
adc_disable(ADC0);
```

adc_calibration_enable

The description of adc_calibration_enable is shown as below:

Table 3-10. Function adc_calibration_enable

Function name	adc_calibration_enable
Function prototype	void adc_calibration_enable(uint32_t adc_periph);
Function descriptions	ADCx calibration and reset calibration
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral

<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* ADC0 calibration and reset calibration */
adc_calibration_enable(ADC0);
```

adc_channel_16_to_18

The description of adc_channel_16_to_18 is shown as below:

Table 3-11. Function adc_channel_16_to_18

Function name	adc_channel_16_to_18
Function prototype	void adc_channel_16_to_18(uint32_t function, ControlStatus newvalue);
Function descriptions	configure temperature sensor and internal reference voltage channel or VBAT channel function
Precondition	-
The called functions	-
Input parameter{in}	
function	temperature sensor and internal reference voltage channel or VBAT channel
<i>ADC_VBAT_CHANNEL_SWITCH</i>	channel 18 (1/4 voltage of external battery) switch of ADC0
<i>ADC_TEMP_VREF_CH_ANNEL_SWITCH</i>	channel 16 (temperature sensor) and 17 (internal reference voltage) switch of ADC0
Input parameter{in}	
newvalue	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable the temperature sensor and Vrefint channel */

adc_channel_16_to_18 (ADC_TEMP_VREF_CHANNEL_SWITCH, ENABLE);
```

adc_resolution_config

The description of adc_resolution_config is shown as below:

Table 3-12. Function adc_resolution_config

Function name	adc_resolution_config
Function prototype	void adc_resolution_config(uint32_t adc_periph , uint32_t resolution);
Function descriptions	configure ADC resolution
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
resolution	ADC resolution
<i>ADC_RESOLUTION_12B</i>	12-bit ADC resolution
<i>ADC_RESOLUTION_10B</i>	10-bit ADC resolution
<i>ADC_RESOLUTION_8B</i>	8-bit ADC resolution
<i>ADC_RESOLUTION_6B</i>	6-bit ADC resolution
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure ADC0 resolution: 10 bits */
adc_resolution_config (ADC0, ADC_RESOLUTION_10B);
```

adc_oversample_mode_config

The description of adc_oversample_mode_config is shown as below:

Table 3-13. Function adc_oversample_mode_config

Function name	adc_oversample_mode_config
Function prototype	void adc_oversample_mode_config(uint32_t adc_periph, uint32_t mode, uint16_t shift, uint8_t ratio);
Function descriptions	configure ADC oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
mode	ADC oversampling mode
ADC_OVERSAMPLING_ALL_CONVERT	all oversampled conversions for a channel are done consecutively after a trigger
ADC_OVERSAMPLING_ONE_CONVERT	each oversampled conversion for a channel needs a trigger
Input parameter{in}	
shift	ADC oversampling shift
ADC_OVERSAMPLING_SHIFT_NONE	no oversampling shift
ADC_OVERSAMPLING_SHIFT_1B	1-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_2B	2-bit oversampling shift

<code>_SHIFT_2B</code>	
<code>ADC_OVERSAMPLING_SHIFT_3B</code>	3-bit oversampling shift
<code>ADC_OVERSAMPLING_SHIFT_4B</code>	4-bit oversampling shift
<code>ADC_OVERSAMPLING_SHIFT_5B</code>	5-bit oversampling shift
<code>ADC_OVERSAMPLING_SHIFT_6B</code>	6-bit oversampling shift
<code>ADC_OVERSAMPLING_SHIFT_7B</code>	7-bit oversampling shift
<code>ADC_OVERSAMPLING_SHIFT_8B</code>	8-bit oversampling shift
Input parameter{in}	
<code>ratio</code>	ADC oversampling ratio
<code>ADC_OVERSAMPLING_RATIO_MUL2</code>	oversampling ratio multiple 2
<code>ADC_OVERSAMPLING_RATIO_MUL4</code>	oversampling ratio multiple 4
<code>ADC_OVERSAMPLING_RATIO_MUL8</code>	oversampling ratio multiple 8
<code>ADC_OVERSAMPLING_RATIO_MUL16</code>	oversampling ratio multiple 16
<code>ADC_OVERSAMPLING_RATIO_MUL32</code>	oversampling ratio multiple 32
<code>ADC_OVERSAMPLING_RATIO_MUL64</code>	oversampling ratio multiple 64
<code>ADC_OVERSAMPLING_RATIO_MUL128</code>	oversampling ratio multiple 128
<code>ADC_OVERSAMPLING_RATIO_MUL256</code>	oversampling ratio multiple 256
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure ADC1 oversample mode: 16 times sample, 4 bits shift */

adc_oversample_mode_config(ADC1, ADC_OVERSAMPLING_ALL_CONVERT,
    ADC_OVERSAMPLING_SHIFT_4B, ADC_OVERSAMPLING_RATIO_MUL16);
```

adc_oversample_mode_enable

The description of adc_oversample_mode_enable is shown as below:

Table 3-14. Function adc_oversample_mode_enable

Function name	adc_oversample_mode_enable
Function prototype	void adc_oversample_mode_enable(uint32_t adc_periph);
Function descriptions	enable ADC oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC0 oversample mode */

adc_oversample_mode_enable (ADC0);
```

adc_oversample_mode_disable

The description of adc_oversample_mode_disable is shown as below:

Table 3-15. Function adc_oversample_mode_disable

Function name	adc_oversample_mode_disable
----------------------	-----------------------------

Function prototype	void adc_oversample_mode_disable(uint32_t adc_periph);
Function descriptions	disable ADC oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 oversample mode */
adc_oversample_mode_disable (ADC0);
```

adc_dma_mode_enable

The description of **adc_dma_mode_enable** is shown as below:

Table 3-16. Function adc_dma_mode_enable

Function name	adc_dma_mode_enable
Function prototype	void adc_dma_mode_enable(uint32_t adc_periph);
Function descriptions	enable ADCx DMA request
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable ADC0 DMA request */

adc_dma_mode_enable(ADC0);
```

adc_dma_mode_disable

The description of adc_dma_mode_disable is shown as below:

Table 3-17. Function adc_dma_mode_disable

Function name	adc_dma_mode_disable
Function prototype	void adc_dma_mode_disable(uint32_t adc_periph);
Function descriptions	disable ADCx DMA request
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 DMA request */

adc_dma_mode_disable(ADC0);
```

adc_dma_request_after_last_enable

The description of adc_dma_request_after_last_enable is shown as below:

Table 3-18. Function adc_dma_request_after_last_enable

Function name	adc_dma_request_after_last_enable
Function prototype	void adc_dma_request_after_last_enable(uint32_t adc_periph);
Function descriptions	when DMA=1, the DMA engine issues a request at end of each regular

	conversion
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* when DMA=1, the DMA engine issues a request at end of each regular conversion for ADC0
 */
adc_dma_request_after_last_enable (ADC0);
```

adc_dma_request_after_last_disable

The description of `adc_dma_request_after_last_disable` is shown as below:

Table 3-19. Function `adc_dma_mode_disable`

Function name	<code>adc_dma_request_after_last_disable</code>
Function prototype	<code>void adc_dma_request_after_last_disable(uint32_t adc_periph);</code>
Function descriptions	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Output parameter{out}	
-	-
Return value	

--	--

Example:

```
/* the DMA engine is disabled after the end of transfer signal from DMA controller is detected
for ADC0 */

adc_dma_request_after_last_disable (ADC0);
```

adc_discontinuous_mode_config

The description of adc_discontinuous_mode_config is shown as below:

Table 3-20. Function adc_discontinuous_mode_config

Function name	adc_discontinuous_mode_config
Function prototype	void adc_discontinuous_mode_config(uint32_t adc_periph, uint8_t adc_channel_group, uint8_t length);
Function descriptions	configure ADC discontinuous mode
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_channel_group	select the channel group
ADC_REGULAR_CHA_NNEL	regular channel group
ADC_INSERTED_CHA_NNEL	inserted channel group
ADC_CHANNEL_DISC_ON_DISABLE	disable discontinuous mode of regular and inserted channel
Input parameter{in}	
length	number of conversions in discontinuous mode, the number can be 1..8 for regular channel, the number has no effect for inserted channel
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure ADC0 regular channel group discontinuous mode */
adc_discontinuous_mode_config(ADC0, ADC_REGULAR_CHANNEL, 6);
```

adc_channel_length_config

The description of adc_channel_length_config is shown as below:

Table 3-21. Function adc_channel_length_config

Function name	adc_channel_length_config
Function prototype	void adc_channel_length_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t length);
Function descriptions	configure the length of regular channel group or inserted channel group
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_channel_group	select the channel group
ADC_REGULAR_CHANNEL	regular channel group
ADC_INSERTED_CHANNEL	inserted channel group
Input parameter{in}	
length	the length of the channel, regular channel 1-16, inserted channel 1-4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure the length of ADC0 regular channel */

adc_channel_length_config(ADC0, ADC_REGULAR_CHANNEL, 4);

```

adc_regular_channel_config

The description of adc_regular_channel_config is shown as below:

Table 3-22. Function adc_regular_channel_config

Function name	adc_regular_channel_config
Function prototype	void adc_regular_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
Function descriptions	configure ADC regular channel
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
rank	the regular group sequence rank, this parameter must be between 0 to 15
Input parameter{in}	
adc_channel	the selected ADC channel
ADC_CHANNEL_x(x=0..18)	ADC Channelx (x=0..17)(x=16..18 are only for ADC0)
Input parameter{in}	
sample_time	the sample time value
ADC_SAMPLETIME_3	3cycles
ADC_SAMPLETIME_15	15cycles
ADC_SAMPLETIME_28	28 cycles
ADC_SAMPLETIME_56	56 cycles

<i>ADC_SAMPLETIME_8</i>	84cycles
<i>4</i>	
<i>ADC_SAMPLETIME_1</i>	112 cycles
<i>12</i>	
<i>ADC_SAMPLETIME_1</i>	144cycles
<i>44</i>	
<i>ADC_SAMPLETIME_4</i>	480 cycles
<i>80</i>	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 regular channel */
adc_regular_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_56);
```

adc_inserted_channel_config

The description of `adc_inserted_channel_config` is shown as below:

Table 3-23. Function `adc_inserted_channel_config`

Function name	adc_inserted_channel_config
Function prototype	void adc_inserted_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
Function descriptions	configure ADC inserted channel
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC peripheral
<code>ADCx(x=0..2)</code>	ADC peripheral selection
Input parameter{in}	
<code>rank</code>	the inserted group sequencer rank, this parameter must be between 0 to 3
Input parameter{in}	

adc_channel	the selected ADC channel
<i>ADC_CHANNEL_x(x=0..18)</i>	ADC Channelx (x=0..17)(x=16..18 are only for ADC0)
Input parameter{in}	
sample_time	the sample time value
<i>ADC_SAMPLETIME_3</i>	3 cycles
<i>ADC_SAMPLETIME_15</i>	15 cycles
<i>ADC_SAMPLETIME_28</i>	28 cycles
<i>ADC_SAMPLETIME_56</i>	56 cycles
<i>ADC_SAMPLETIME_48</i>	48 cycles
<i>ADC_SAMPLETIME_112</i>	112 cycles
<i>ADC_SAMPLETIME_144</i>	144 cycles
<i>ADC_SAMPLETIME_480</i>	480 cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 inserted channel */
adc_inserted_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_56);
```

adc_inserted_channel_offset_config

The description of `adc_inserted_channel_offset_config` is shown as below:

Table 3-24. Function `adc_inserted_channel_offset_config`

Function name	adc_inserted_channel_offset_config

Function prototype	void adc_inserted_channel_offset_config(uint32_t adc_periph, uint8_t inserted_channel, uint16_t offset);
Function descriptions	configure ADC inserted channel offset
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
inserted_channel	insert channel select
<i>ADC_INSERTED_CHANNEL_x(x=0..3)</i>	inserted channel, x=0,1,2,3
Input parameter{in}	
offset	the offset data, this parameter must be between 0 to 4095
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 inserted channel offset */
adc_inserted_channel_offset_config(ADC0, ADC_INSERTED_CHANNEL_0, 100);
```

adc_external_trigger_source_config

The description of `adc_external_trigger_source_config` is shown as below:

Table 3-25. Function `adc_external_trigger_source_config`

Function name	adc_external_trigger_source_config
Function prototype	void adc_external_trigger_source_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t external_trigger_source);
Function descriptions	configure ADC external trigger source
Precondition	-

The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC peripheral
<code>ADCx(x=0..2)</code>	ADC peripheral selection
Input parameter{in}	
<code>adc_channel_group</code>	select the channel group
<code>ADC_REGULAR_CHA_NNEL</code>	regular channel group
<code>ADC_INSERTED_CHA_NNEL</code>	inserted channel group
Input parameter{in}	
<code>external_trigger_source</code>	regular or inserted group trigger source
<code>ADC_EXTRIG_REGULAR_T0_CH0</code>	external trigger timer 0 CC0 event select for regular channel
<code>ADC_EXTRIG_REGULAR_T0_CH1</code>	external trigger timer 0 CC1 event select for regular channel
<code>ADC_EXTRIG_REGULAR_T0_CH2</code>	external trigger timer 7 CC2 event select for regular channel
<code>ADC_EXTRIG_REGULAR_T1_CH1</code>	external trigger timer 1 CC1 event select for regular channel
<code>ADC_EXTRIG_REGULAR_T1_CH2</code>	external trigger timer 1 CC2 event select for regular channel
<code>ADC_EXTRIG_REGULAR_T1_CH3</code>	external trigger timer 1 CC3 event select for regular channel
<code>ADC_EXTRIG_REGULAR_T1_TRGO</code>	external trigger timer 1 TRGO event select for regular channel
<code>ADC_EXTRIG_REGULAR_T2_CH0</code>	external trigger timer 2 CC0 event select for regular channel
<code>ADC_EXTRIG_REGULAR_T2_TRGO</code>	external trigger timer 2 TRGO event select for regular channel
<code>ADC_EXTRIG_REGULAR_T3_CH3</code>	external trigger timer 3 CC3 event select for regular channel

<i>ADC_EXTTRIG_REGU_LAR_T4_CH0</i>	external trigger timer 4 CC0 event select for regular channel
<i>ADC_EXTTRIG_REGU_LAR_T4_CH1</i>	external trigger timer 4 CC1 event select for regular channel
<i>ADC_EXTTRIG_REGU_LAR_T4_CH2</i>	external trigger timer 4 CC2 event select for regular channel
<i>ADC_EXTTRIG_REGU_LAR_T7_CH0</i>	external trigger timer 7 CC0 event select for regular channel
<i>ADC_EXTTRIG_REGU_LAR_T7_TRGO</i>	external trigger timer 7 TRGO event select for regular channel
<i>ADC_EXTTRIG_REGU_LAR_EXTI_11</i>	external trigger extiline 11 select for regular channel
<i>ADC_EXTTRIG_INSER TED_T0_CH3</i>	external trigger timer 0 CH3 event select for inserted channel
<i>ADC_EXTTRIG_INSER TED_T0_TRGO</i>	external trigger timer 0 TRGO event select for inserted channel
<i>ADC_EXTTRIG_INSER TED_T1_CH0</i>	external trigger timer 1 CH0 event select for inserted channel
<i>ADC_EXTTRIG_INSER TED_T1_TRGO</i>	external trigger timer 1 TRGO event select for inserted channel
<i>ADC_EXTTRIG_INSER TED_T2_CH1</i>	external trigger timer 2 CH1 event select for inserted channel
<i>ADC_EXTTRIG_INSER TED_T2_CH3</i>	external trigger timer 2 CH3 event select for inserted channel
<i>ADC_EXTTRIG_INSER TED_T3_CH0</i>	external trigger timer 3 CH0 event select for inserted channel
<i>ADC_EXTTRIG_INSER TED_T3_CH1</i>	external trigger timer 3 CH1 event select for inserted channel
<i>ADC_EXTTRIG_INSER TED_T3_CH2</i>	external trigger timer 3 CH2 event select for inserted channel
<i>ADC_EXTTRIG_INSER TED_T3_TRGO</i>	external trigger timer 3 TRGO event select for inserted channel
<i>ADC_EXTTRIG_INSER TED_T4_CH3</i>	external trigger timer 4 CH3 event select for inserted channel

<i>ADC_EXTTRIG_INSERTED_TED_T4_TRGO</i>	external trigger timer 4 TRGO event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_TED_T7_CH1</i>	external trigger timer 7 CH1 event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_TED_T7_CH2</i>	external trigger timer 7 CH2 event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_TED_T7_CH3</i>	external trigger timer 7 CH3 event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_TED EXTI_15</i>	external trigger extiline 15 select for inserted channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure ADC0 regular channel external trigger source */

adc_external_trigger_source_config(ADC0,ADC_REGULAR_CHANNEL,
ADC_EXTTRIG_REGULAR_T0_CH0);

```

adc_external_trigger_config

The description of **adc_external_trigger_config** is shown as below:

Table 3-26. Function adc_external_trigger_config

Function name	adc_external_trigger_config
Function prototype	void adc_external_trigger_config(uint32_t adc_periph , uint8_t adc_channel_group , uint32_t trigger_mode);
Function descriptions	configure ADC external trigger
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection

Input parameter{in}	
adc_channel_group	select the channel group
<i>ADC_REGULAR_CHANNEL</i>	regular channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
Input parameter{in}	
trigger_mode	external trigger mode
<i>EXTERNAL_TRIGGER_DISABLE</i>	external trigger disable
<i>EXTERNAL_TRIGGER_RISING</i>	rising edge of external trigger
<i>EXTERNAL_TRIGGER_FALLING</i>	falling edge of external trigger
<i>EXTERNAL_TRIGGER_RISING_FALLING</i>	rising and falling edge of external trigger
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC0 inserted channel group external trigger */
adc_external_trigger_config(ADC0, ADC_INSERTED_CHANNEL_0,
EXTERNAL_TRIGGER_RISING);
```

adc_software_trigger_enable

The description of `adc_software_trigger_enable` is shown as below:

Table 3-27. Function `adc_software_trigger_enable`

Function name	adc_software_trigger_enable
Function prototype	void adc_software_trigger_enable(uint32_t adc_periph, uint8_t adc_channel_group);
Function descriptions	enable ADC software trigger

Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
adc_channel_group	select the channel group
<i>ADC_REGULAR_CHA_NNEL</i>	regular channel group
<i>ADC_INSERTED_CHA_NNEL</i>	inserted channel group
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC0 regular channel group software trigger */
adc_software_trigger_enable(ADC0, ADC_REGULAR_CHANNEL);
```

adc_end_of_conversion_config

The description of `adc_end_of_conversion_config` is shown as below:

Table 3-28. Function `adc_end_of_conversion_config`

Function name	adc_end_of_conversion_config
Function prototype	void adc_end_of_conversion_config(uint32_t adc_periph , uint8_t end_selection);
Function descriptions	configure end of conversion mode
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral

<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
end_selection	end of conversion mode
<i>ADC_EOC_SET_SEQ_UENCE</i>	only at the end of a sequence of regular conversions, the EOC bit is set.Overflow detection is disabled unless DMA=1
<i>ADC_EOC_SET_CONVERSION</i>	at the end of each regular conversion, the EOC bit is set.Overflow is detected automatically
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* at the end of each regular conversion, the EOC bit is set. */
adc_end_of_conversion_config (ADC0, ADC_EOC_SET_CONVERSION);
```

adc_regular_data_read

The description of adc_regular_data_read is shown as below:

Table 3-29. Function adc_regular_data_read

Function name	adc_regular_data_read
Function prototype	uint16_t adc_regular_data_read(uint32_t adc_periph);
Function descriptions	read ADC regular group data register
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
uint16_t	ADC conversion value (0-0xFFFF)

Example:

```
/* read ADC0 regular group data register */

uint16_t adc_value = 0;

adc_value = adc_regular_data_read(ADC0);
```

adc_inserted_data_read

The description of `adc_inserted_data_read` is shown as below:

Table 3-30. Function `adc_inserted_data_read`

Function name	adc_inserted_data_read
Function prototype	uint16_t adc_inserted_data_read(uint32_t adc_periph, uint8_t inserted_channel);
Function descriptions	read ADC inserted group data register
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC peripheral
<code>ADCx(x=0..2)</code>	ADC peripheral selection
Input parameter{in}	
<code>inserted_channel</code>	insert channel select
<code>ADC_INSERTED_CHANNEL_x(x=0..3)</code>	inserted Channelx, x=0,1,2,3
Output parameter{out}	
-	-
Return value	
<code>uint16_t</code>	ADC conversion value (0-0xFFFF)

Example:

```
/* read ADC0 inserted group data register */

uint16_t adc_value = 0;

adc_value = adc_inserted_data_read (ADC0, ADC_INSERTED_CHANNEL_0);
```

adc_watchdog_single_channel_disable

The description of adc_watchdog_single_channel_disable is shown as below:

Table 3-31. Function adc_watchdog_single_channel_disable

Function name	adc_watchdog_single_channel_disable
Function prototype	void adc_watchdog_single_channel_disable(uint32_t adc_periph);
Function descriptions	disable ADC analog watchdog single channel
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 analog watchdog single channel */
adc_watchdog_single_channel_disable(ADC0);
```

adc_watchdog_single_channel_enable

The description of adc_watchdog_single_channel_enable is shown as below:

Table 3-32. Function adc_watchdog_single_channel_enable

Function name	adc_watchdog_single_channel_enable
Function prototype	void adc_watchdog_single_channel_enable(uint32_t adc_periph, uint8_t adc_channel);
Function descriptions	configure ADC analog watchdog single channel
Precondition	-
The called functions	-
Input parameter{in}	

adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
adc_channel	the selected ADC channel
<i>ADC_CHANNEL_x(x=0..17)</i>	ADC Channelx(x=0..17) (x=16 and x=17 are only for ADC0)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 analog watchdog single channel */
adc_watchdog_single_channel_enable(ADC0, ADC_CHANNEL_1);
```

adc_watchdog_group_channel_enable

The description of `adc_watchdog_group_channel_enable` is shown as below:

Table 3-33. Function `adc_watchdog_group_channel_enable`

Function name	adc_watchdog_group_channel_enable
Function prototype	void adc_watchdog_group_channel_enable(uint32_t adc_periph, uint8_t adc_channel_group);
Function descriptions	configure ADC analog watchdog group channel
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
adc_channel_group	the channel group use analog watchdog
<i>ADC_REGULAR_CHANNEL</i>	regular channel group

<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
<i>ADC_REGULAR_INSERTED_CHANNEL</i>	both regular and inserted group
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 analog watchdog group channel */
adc_watchdog_group_channel_enable(ADC0, ADC_REGULAR_CHANNEL);
```

adc_watchdog_disable

The description of `adc_watchdog_disable` is shown as below:

Table 3-34. Function `adc_watchdog_disable`

Function name	adc_watchdog_disable
Function prototype	void adc_watchdog_disable(uint32_t adc_periph);
Function descriptions	disable ADC analog watchdog
Precondition	-
The called functions	-
Input parameter{in}	
<i>adc_periph</i>	ADC peripheral
<i>ADCx(0,1)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 analog watchdog */
adc_watchdog_disable(ADC0);
```

adc_watchdog_threshold_config

The description of `adc_watchdog_threshold_config` is shown as below:

Table 3-35. Function `adc_watchdog_threshold_config`

Function name	adc_watchdog_threshold_config
Function prototype	void adc_watchdog_threshold_config(uint32_t adc_periph, uint16_t low_threshold, uint16_t high_threshold);
Function descriptions	configure ADC analog watchdog threshold
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC peripheral
<code>ADCx(x=0..2)</code>	ADC peripheral selection
Input parameter{in}	
<code>low_threshold</code>	analog watchdog low threshold, 0..4095
Input parameter{in}	
<code>high_threshold</code>	analog watchdog high threshold, 0..4095
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure ADC0 analog watchdog threshold */

adc_watchdog_threshold_config(ADC0, 0x0400, 0x0A00);

```

adc_flag_get

The description of `adc_flag_get` is shown as below:

Table 3-36. Function `adc_flag_get`

Function name	adc_flag_get
Function prototype	FlagStatus adc_flag_get(uint32_t adc_periph , uint32_t adc_flag);

Function descriptions	get the ADC flag bits
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_flag	the adc flag bits
ADC_FLAG_WDE	analog watchdog event flag
ADC_FLAG_EOC	end of group conversion flag
ADC_FLAG_EOIC	end of inserted group conversion flag
ADC_FLAG_STIC	start flag of inserted channel group
ADC_FLAG_STRC	start flag of regular channel group
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the ADC0 analog watchdog flag bits*/
FlagStatus flag_value;
flag_value = adc_flag_get(ADC0, ADC_FLAG_WDE);
```

adc_flag_clear

The description of `adc_flag_clear` is shown as below:

Table 3-37. Function `adc_flag_clear`

Function name	adc_flag_clear
Function prototype	void adc_flag_clear(uint32_t adc_periph, uint32_t adc_flag);
Function descriptions	clear the ADC flag bits
Precondition	-

The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_flag	the adc flag bits
ADC_FLAG_WDE	analog watchdog event flag
ADC_FLAG_EOC	end of group conversion flag
ADC_FLAG_EOIC	end of inserted group conversion flag
ADC_FLAG_STIC	start flag of inserted channel group
ADC_FLAG_STRC	start flag of regular channel group
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the ADC0 analog watchdog flag bits*/
adc_flag_clear(ADC0, ADC_FLAG_WDE);
```

adc_regular_software_startconv_flag_get

The description of `adc_regular_software_startconv_flag_get` is shown as below:

Table 3-38. Function `adc_regular_software_startconv_flag_get`

Function name	adc_regular_software_startconv_flag_get
Function prototype	FlagStatus adc_regular_software_startconv_flag_get(uint32_t adc_periph);
Function descriptions	get the bit state of ADCx software regular channel start conversion
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral

<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the bit state of ADC0 software regular channel start conversion */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_regular_software_startconv_flag_get(ADC0);
```

adc_inserted_software_startconv_flag_get

The description of `adc_inserted_software_startconv_flag_get` is shown as below:

Table 3-39. Function `adc_inserted_software_startconv_flag_get`

Function name	adc_inserted_software_startconv_flag_get
Function prototype	FlagStatus adc_inserted_software_startconv_flag_get(uint32_t adc_periph);
Function descriptions	get the bit state of ADCx software inserted channel start conversion
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the bit state of ADC0 software inserted channel start conversion */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_inserted_software_startconv_flag_get(ADC0);
```

adc_interrupt_flag_get

The description of adc_interrupt_flag_get is shown as below:

Table 3-40. Function adc_interrupt_flag_get

Function name	adc_interrupt_flag_get
Function prototype	FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, uint32_t adc_interrupt);
Function descriptions	get the ADC interrupt bits
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_interrupt	the adc interrupt bits
ADC_INT_WDE	analog watchdog interrupt
ADC_INT_EOC	end of group conversion interrupt
ADC_INT_EOIC	end of inserted group conversion interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the ADC0 analog watchdog interrupt bits*/
FlagStatus flag_value;
flag_value = adc_interrupt_flag_get(ADC0, ADC_INT_WDE);
```

adc_interrupt_flag_clear

The description of adc_interrupt_flag_clear is shown as below:

Table 3-41. Function adc_interrupt_flag_clear

Function name	adc_interrupt_flag_clear
Function prototype	void adc_interrupt_flag_clear(uint32_t adc_periph, uint32_t adc_interrupt);
Function descriptions	clear the ADC interrupt bits
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_interrupt	the adc interrupt bits
ADC_INT_WDE	analog watchdog interrupt
ADC_INT_EOC	end of group conversion interrupt
ADC_INT_EOIC	end of inserted group conversion interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the ADC0 analog watchdog interrupt bits*/
adc_interrupt_flag_clear(ADC0, ADC_INT_WDE);
```

adc_interrupt_enable

The description of adc_interrupt_enable is shown as below:

Table 3-42. Function adc_interrupt_enable

Function name	adc_interrupt_enable
Function prototype	void adc_interrupt_enable(uint32_t adc_periph, uint32_t adc_interrupt);
Function descriptions	enable ADC interrupt
Precondition	-

The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_interrupt	the adc interrupt
ADC_INT_WDE	analog watchdog interrupt
ADC_INT_EOC	end of group conversion interrupt
ADC_INT_EOIC	end of inserted group conversion interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC0 analog watchdog interrupt */
adc_interrupt_enable(ADC0, ADC_INT_WDE);
```

adc_interrupt_disable

The description of **adc_interrupt_disable** is shown as below:

Table 3-43. Function adc_interrupt_disable

Function name	adc_interrupt_disable
Function prototype	void adc_interrupt_disable(uint32_t adc_periph , uint32_t adc_interrupt);
Function descriptions	Disable ADC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	

adc_interrupt	the adc interrupt
<i>ADC_INT_WDE</i>	analog watchdog interrupt
<i>ADC_INT_EOC</i>	end of group conversion interrupt
<i>ADC_INT_EOIC</i>	end of inserted group conversion interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 interrupt */

adc_interrupt_disable(ADC0, ADC_INT_WDE);
```

adc_sync_mode_config

The description of **adc_sync_mode_config** is shown as below:

Table 3-44. Function `adc_sync_mode_config`

Function name	adc_sync_mode_config
Function prototype	void adc_sync_mode_config(uint32_t sync_mode);
Function descriptions	configure the ADC sync mode
Precondition	-
The called functions	-
Input parameter{in}	
sync_mode	ADC sync mode
<i>ADC_SYNC_MODE_IN_DEPENDENT</i>	all the ADCs work independently
<i>ADC_DAUL_REGULAR_PARALLEL_INSERTED_D_PARALLEL</i>	ADC0 and ADC1 work in combined regular parallel & inserted parallel mode
<i>ADC_DAUL_REGULAR_PARALLEL_INSERTED_D_ROTATION</i>	ADC0 and ADC1 work in combined regular parallel & trigger rotation mode
<i>ADC_DAUL_INSERTED</i>	ADC0 and ADC1 work in inserted parallel mode

<i>D_PARALLEL</i>	
<i>ADC_DAUL_REGULAR_PARALLEL</i>	ADC0 and ADC1 work in regular parallel mode
<i>ADC_DAUL_REGULAR_FOLLOW_UP</i>	ADC0 and ADC1 work in follow-up mode
<i>ADC_DAUL_INSERTED_TRIGGER_ROTATION</i>	ADC0 and ADC1 work in trigger rotation mode
<i>ADC_ALL_REGULAR_PARALLEL_INSERTED_PARALLEL</i>	all ADCs work in combined regular parallel & inserted parallel mode
<i>ADC_ALL_REGULAR_PARALLEL_INSERTED_ROTATION</i>	all ADCs work in combined regular parallel & trigger rotation mode
<i>ADC_ALL_INSERTED_PARALLEL</i>	all ADCs work in inserted parallel mode
<i>ADC_ALL_REGULAR_PARALLEL</i>	all ADCs work in regular parallel mode
<i>ADC_ALL_REGULAR_FOLLOW_UP</i>	all ADCs work in follow-up mode
<i>ADC_ALL_INSERTED_TRIGGER_ROTATION</i>	all ADCs work in trigger rotation mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* ADC0 and ADC1 work in combined regular parallel & inserted parallel mode */

adc_sync_mode_config (ADC_DAUL_REGULAR_PARALLEL_INSERTED_PARALLEL);
```

adc_sync_delay_config

The description of `adc_sync_delay_config` is shown as below:

Table 3-45. Function adc_interrupt_disable

Function name	adc_sync_delay_config
Function prototype	void adc_sync_delay_config(uint32_t sample_delay);
Function descriptions	configure the delay between 2 sampling phases in ADC sync modes
Precondition	-
The called functions	-
Input parameter{in}	
sample_delay	the delay between 2 sampling phases in ADC sync modes
<i>ADC_SYNC_DELAY_x CYCLE(x=5..20)</i>	the delay between 2 sampling phases in ADC sync modes is x ADC clock cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the delay between 2 sampling phases in ADC sync modes */
adc_sync_delay_config (ADC_SYNC_DELAY_5CYCLE);
```

adc_sync_dma_config

The description of adc_sync_dma_config is shown as below:

Table 3-46. Function adc_sync_dma_config

Function name	adc_sync_dma_config
Function prototype	void adc_sync_dma_config(uint32_t dma_mode);
Function descriptions	configure ADC sync DMA mode selection
Precondition	-
The called functions	-
Input parameter{in}	
dma_mode	the adc interrupt
<i>ADC_SYNC_DMA_DISABLE</i>	ADC sync DMA disabled

<i>ADC_SYNC_DMA_MODE0</i>	ADC sync DMA mode 0
<i>ADC_SYNC_DMA_MODE1</i>	ADC sync DMA mode 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC sync DMA mode selection */
adc_sync_dma_config (ADC_SYNC_DMA_MODE0);
```

adc_sync_dma_request_after_last_enable

The description of `adc_sync_dma_request_after_last_enable` is shown as below:

Table 3-47. Function `adc_sync_dma_request_after_last_enable`

Function name	adc_sync_dma_request_after_last_enable
Function prototype	void adc_sync_dma_request_after_last_enable(void);
Function descriptions	when SYNC DMA is not equal to 2'b00, the DMA engine issues requests according to the SYNC DMA bits
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* when SYNC DMA is not equal to 2'b00, the DMA engine issues requests according to the
SYNC DMA bits */
```

```
adc_sync_dma_request_after_last_enable();
```

adc_sync_dma_request_after_last_disable

The description of adc_sync_dma_request_after_last_disable is shown as below:

Table 3-48. Function adc_sync_dma_request_after_last_disable

Function name	adc_sync_dma_request_after_last_disable
Function prototype	void adc_sync_dma_request_after_last_disable (void);
Function descriptions	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* the DMA engine is disabled after the end of transfer signal from DMA controller is detected
*/
adc_sync_dma_request_after_last_disable();
```

adc_sync_regular_data_read

The description of adc_sync_regular_data_read is shown as below:

Table 3-49. Function adc_interrupt_disable

Function name	adc_sync_regular_data_read
Function prototype	uint32_t adc_sync_regular_data_read(void);
Function descriptions	read ADC sync regular data register
Precondition	-
The called functions	-
Input parameter{in}	

-	-
Output parameter{out}	
-	-
Return value	
uint32_t	ADC sync regular data register

Example:

```
/* read ADC sync regular data register */

adc_sync_regular_data_read();
```

3.3. CAN

CAN bus (for Controller Area Network) is a bus standard designed to allow microcontrollers and devices to communicate with each other without a host computer. The CAN registers are listed in chapter [3.3.1](#), the CAN firmware functions are introduced in chapter [3.3.2](#).

3.3.1. Descriptions of Peripheral registers

CAN registers are listed in the table shown as below:

Table 3-50. CAN Registers

Registers	Descriptions
CAN_CTL	Control register
CAN_STAT	Status register
CAN_TSTAT	Transmit status register
CAN_RFIFO0	Receive message FIFO0 register
CAN_RFIFO1	Receive message FIFO1 register
CAN_INTEN	Interrupt enable register
CAN_ERR	Error register
CAN_BT	Bit timing register
CAN_TMIx	Transmit mailbox identifier register
CAN_TMPx	Transmit mailbox property register
CAN_TMDATA0x	Transmit mailbox data0 register

Registers	Descriptions
CAN_TMDATA1x	Transmit mailbox data1 register
CAN_RFIFOMIx	Receive FIFO mailbox identifier register
CAN_RFIFOMPx	Receive FIFO mailbox property register
CAN_RFIFOMDAT A0x	Receive FIFO mailbox data0 register
CAN_RFIFOMDAT A1x	Receive FIFO mailbox data1 register
CAN_FCTL	Filter control register
CAN_FMCFG	Filter mode configuration register
CAN_FSCFG	Filter scale configuration register
CAN_FA FIFO	Filter associated FIFO register
CAN_FW	Filter working register
CAN_FxDATAy	Filter x data y register

3.3.2. Descriptions of Peripheral functions

CAN firmware functions are listed in the table shown as below:

Table 3-51. CAN firmware function

Function name	Function description
can_deinit	deinitialize CAN
can_struct_para_init	initialize CAN parameter struct with a default value
can_init	initialize CAN
can_filter_init	initialize CAN filter
can1_filter_start_bank	set can1 filter start bank number
can_debug_freeze_enable	CAN debug freeze enable
can_debug_freeze_disable	CAN debug freeze disable
can_time_trigger_mode_enable	CAN time trigger mode enable
can_time_trigger_mode_disable	CAN time trigger mode disable
can_message_transmit	transmit CAN message

Function name	Function description
can_transmit_states	get CAN transmit state
can_transmission_stop	stop CAN transmission
can_message_receive	CAN receive message
can_fifo_release	CAN release fifo
can_receive_message_length_get	CAN receive message length
can_working_mode_set	CAN working mode
can_wakeup	CAN wakeup from sleep mode
can_error_get	CAN get error
can_receive_error_number_get	get CAN receive error number
can_transmit_error_number_get	get CAN transmit error number
can_interrupt_enable	CAN interrupt enable
can_interrupt_disable	CAN interrupt disable
can_flag_get	CAN get flag state
can_flag_clear	CAN clear flag state
can_interrupt_flag_get	CAN get interrupt flag state
can_interrupt_flag_clear	CAN clear interrupt flag state

Structure can_parameter_struct

Table 3-52. Structure can_parameter_struct

Member name	Function description
working_mode	CAN working mode
resync_jump_width	CAN resynchronization jump width
time_segment_1	time segment 1
time_segment_2	time segment 2
time_triggered	time triggered communication mode
auto_bus_off_recovery	automatic bus-off recovery
auto_wake_up	automatic wake-up mode

auto_retrans	automatic retransmission mode
rec_fifo_overwrite	receive FIFO overwrite mode
trans_fifo_order	transmit FIFO order
prescaler	baudrate prescaler

Structure can_trasnmit_message_struct

Table 3-53. Structure can_trasnmit_message_struct

Member name	Function description
tx_sfid	standard format frame identifier
tx_efid	extended format frame identifier
tx_ff	format of frame, standard or extended format
tx_ft	type of frame, data or remote
tx_dlen	data length
tx_data[8]	transmit data

Structure can_receive_message_struct

Table 3-54. Structure can_receive_message_struct

Member name	Function description
rx_sfid	standard format frame identifier
rx_efid	extended format frame identifier
rx_ff	format of frame, standard or extended format
rx_ft	type of frame, data or remote
rx_dlen	data length
rx_data[8]	receive data
rx_hi	filtering index

Structure can_filter_parameter_struct

Table 3-55. Structure can_filter_parameter_struct

Member name	Function description
filter_list_high	filter list number high bits

filter_list_low	filter list number low bits
filter_mask_high	filter mask number high bits
filter_mask_low	filter mask number low bits
filter_fifo_number	receive FIFO associated with the filter
filter_number	filter number
filter_mode	filter mode, list or mask
filter_bits	filter scale
filter_enable	filter work or not

can_deinit

The description of can_deinit is shown as below:

Table 3-56. Function can_deinit

Function name	can_deinit
Function prototype	void can_deinit(uint32_t can_periph);
Function descriptions	deinitialize CAN
Precondition	-
The called functions	rcu_periph_reset_enable/ rcu_periph_reset_disable
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-
Function name	can_deinit
Function prototype	void can_deinit(uint32_t can_periph);
Function descriptions	deinitialize CAN
Precondition	-
The called functions	rcu_periph_reset_enable/ rcu_periph_reset_disable

Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CAN0 deinitialize*/
can_deinit (CAN0);
```

can_struct_para_init

The description of can_struct_para_init is shown as below:

Table 3-57. Function can_struct_para_init

Function name	can_struct_para_init
Function prototype	void can_struct_para_init(can_struct_type_enum type, void* p_struct)
Function descriptions	initialize CAN parameter struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
type	CAN peripheral
CAN_INIT_STRUCT	CAN initiliaze parameters struct
CAN_FILTER_STRUCT	CAN filter parameters struct
CAN_TX_MESSAGE_STRUCT	CAN transmit message struct
CAN_RX_MESSAGE_STRUCT	CAN receive message struct
Output parameter{out}	
p_struct	the struct pointer that needs initialize

Return value	
-	-

Example:

```
can_parameter_struct can_init;
can_struct_para_init(CAN_INIT_STRUCT, &can_init);
```

can_init

The description of can_init is shown as below:

Table 3-58. Function can_init

Function name	can_init
Function prototype	ErrStatus can_init(uint32_t can_periph, can_parameter_struct* can_parameter_init);
Function descriptions	initialize CAN
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
can_parameter_init	CAN parameter initialization stuct, the structure members can refer to members of the structure Table 3-52. Structure can_parameter_struct
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR
Function name	can_init
Function prototype	ErrStatus can_init(uint32_t can_periph, can_parameter_struct* can_parameter_init);
Function descriptions	initialize CAN

Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
can_parameter_init	CAN parameter initialization stuct, the structure members can refer to members of the structure <u>Table 3-52. Structure can_parameter_struct</u>
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR

Example:

```
/* CAN0 initialize*/
can_init (CAN0);
```

can_filter_init

The description of can_filter_init is shown as below:

Table 3-59. Function can_filter_init

Function name	can_filter_init
Function prototype	void can_filter_init(can_filter_parameter_struct* can_filter_parameter_init);
Function descriptions	initialize CAN filter
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_filter_parameter_init	CAN filter initialization stuct, the structure members can refer to members of the structure <u>Table 3-55. Structure can_filter_parameter_struct</u>
Output parameter{out}	
-	-

Return value	
-	-
Function name	can_filter_init
Function prototype	void can_filter_init(can_filter_parameter_struct* can_filter_parameter_init);
Function descriptions	initialize CAN filter
Precondition	-
The called functions	-
Input parameter{in}	
can_filter_parameter_i nit	CAN filter initialization stuct, the structure members can refer to members of the structure Table 3-55. Structure can_filter_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize CAN filter */
can_filter_init(&can_filter);
```

can1_filter_start_bank

The description of can1_filter_start_bank is shown as below:

Table 3-60. Function can1_filter_start_bank

Function name	can1_filter_start_bank
Function prototype	void can1_filter_start_bank(uint8_t start_bank);
Function descriptions	set CAN1 fliter start bank number
Precondition	-
The called functions	-
Input parameter{in}	
start_bank	CAN1 start bank number
1..27	start number

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set CAN1 filter start bank number 15*/
can1_filter_start_bank (15);
```

can_debug_freeze_enable

The description of can_debug_freeze_enable is shown as below:

Table 3-61. Function can_debug_freeze_enable

Function name	can_debug_freeze_enable
Function prototype	void can_debug_freeze_enable(uint32_t can_periph);
Function descriptions	enable CAN debug freeze
Precondition	-
The called functions	dbg_periph_enable
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-
Function name	can_debug_freeze_enable
Function prototype	void can_debug_freeze_enable(uint32_t can_periph);
Function descriptions	enable CAN debug freeze
Precondition	-
The called functions	dbg_periph_enable
Input parameter{in}	

can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable CAN0 debug freeze */
can_debug_freeze_enable (CAN0);
```

can_debug_freeze_disable

The description of can_debug_freeze_disable is shown as below:

Table 3-62. Function can_debug_freeze_disable

Function name	can_debug_freeze_disable
Function prototype	void can_debug_freeze_disable(uint32_t can_periph);
Function descriptions	disable CAN debug freeze
Precondition	-
The called functions	dbg_periph_disable
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-
Function name	can_debug_freeze_disable
Function prototype	void can_debug_freeze_disable(uint32_t can_periph);
Function descriptions	disable CAN debug freeze
Precondition	-

The called functions	dbg_periph_disable
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable CAN0 debug freeze */
can_debug_freeze_disable (CAN0);
```

can_time_trigger_mode_enable

The description of can_time_trigger_mode_enable is shown as below:

Table 3-63. Function can_time_trigger_mode_enable

Function name	can_time_trigger_mode_enable
Function prototype	void can_time_trigger_mode_enable(uint32_t can_periph);
Function descriptions	enable CAN time trigger mode
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-
Function name	can_time_trigger_mode_enable
Function prototype	void can_time_trigger_mode_enable(uint32_t can_periph);

Function descriptions	enable CAN time trigger mode
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable CAN0 time trigger mode */
can_time_trigger_mode_enable (CAN0);
```

can_time_trigger_mode_disable

The description of can_time_trigger_mode_disable is shown as below:

Table 3-64. Function can_time_trigger_mode_disable

Function name	can_time_trigger_mode_disable
Function prototype	void can_time_trigger_mode_disable(uint32_t can_periph);
Function descriptions	disable CAN time trigger mode
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Function name	can_time_trigger_mode_disable
Function prototype	void can_time_trigger_mode_disable(uint32_t can_periph);
Function descriptions	disable CAN time trigger mode
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable CAN0 time trigger mode */
can_time_trigger_mode_disable (CAN0);
```

can_message_transmit

The description of can_message_transmit is shown as below:

Table 3-65. Function can_message_transmit

Function name	can_message_transmit
Function prototype	uint8_t can_message_transmit(uint32_t can_periph, can_transmit_message_struct* transmit_message);
Function descriptions	transmit CAN message
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
Input parameter{in}	

transmit_message	CAN transmit message stuct, the structure members can refer to members of the structure <u>Table 3-53. Structure can_trasnmit_message_struct</u>
Output parameter{out}	
-	-
Return value	
uint8_t	0x00-0x03
Function name	can_message_transmit
Function prototype	uint8_t can_message_transmit(uint32_t can_periph, can_trasnmit_message_struct* transmit_message);
Function descriptions	transmit CAN message
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
Input parameter{in}	
transmit_message	CAN transmit message stuct, the structure members can refer to members of the structure <u>Table 3-53. Structure can_trasnmit_message_struct</u>
Output parameter{out}	
-	-
Return value	
uint8_t	0x00-0x03

Example:

```
/* CAN0 transmit message and return the mailbox number*/
uint8_t transmit_mailbox = 0;
transmit_mailbox = can_message_transmit(CAN0, &transmit_message);
```

can_transmit_states

The description of can_transmit_states is shown as below:

Table 3-66. Function can_transmit_states

Function name	can_transmit_states
Function prototype	can_transmit_state_enum can_transmit_states(uint32_t can_periph, uint8_t mailbox_number);
Function descriptions	get CAN transmit state
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
mailbox_number	Mailbox number
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
Output parameter{out}	
-	-
Return value	
can_transmit_state_enum	0..4
Function name	can_transmit_states
Function prototype	can_transmit_state_enum can_transmit_states(uint32_t can_periph, uint8_t mailbox_number);
Function descriptions	get CAN transmit state
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
mailbox_number	Mailbox number
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)

Output parameter{out}	
-	-
Return value	
can_transmit_state_e num	0..4

Example:

```
/* CAN0 mailbox0 transmit state */

uint8_t transmit_state = 0;

transmit_state = can_transmit_states (CAN0, CAN_MAILBOX0);
```

can_transmission_stop

The description of can_transmission_stop is shown as below:

Table 3-67. Function can_transmission_stop

Function name	can_transmission_stop
Function prototype	void can_transmission_stop(uint32_t can_periph, uint8_t mailbox_number);
Function descriptions	stop CAN transmission
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
Input parameter{in}	
mailbox_number	Mailbox number
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-
Function name	can_transmission_stop

Function prototype	void can_transmission_stop(uint32_t can_periph, uint8_t mailbox_number);
Function descriptions	stop CAN transmission
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
mailbox_number	Mailbox number
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* stop CAN0 mailbox0 transmission */

can_transmission_stop (CAN0, CAN_MAILBOX0);

```

can_message_receive

The description of can_message_receive is shown as below:

Table 3-68. Function can_message_receive

Function name	can_message_receive
Function prototype	void can_message_receive(uint32_t can_periph, uint8_t fifo_number, can_receive_message_struct* receive_message);
Function descriptions	CAN receive message
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral

<code>CANx(x=0, 1)</code>	CAN peripheral selection
Input parameter{in}	
<code>fifo_number</code>	Fifo number
<code>CAN_FIFOx</code>	<code>CAN_FIFOx(x=0,1)</code>
Input parameter{in}	
<code>receive_message</code>	CAN message receive stuct, the structure members can refer to members of the structure Table 3-54. Structure can receive message struct
Output parameter{out}	
-	-
Return value	
-	-
Function name	<code>can_message_receive</code>
Function prototype	<code>void can_message_receive(uint32_t can_periph, uint8_t fifo_number, can_receive_message_struct* receive_message);</code>
Function descriptions	CAN receive message
Precondition	-
The called functions	-
Input parameter{in}	
<code>can_periph</code>	CAN peripheral
<code>CANx(x=0, 1)</code>	CAN peripheral selection
Input parameter{in}	
<code>fifo_number</code>	Fifo number
<code>CAN_FIFOx</code>	<code>CAN_FIFOx(x=0,1)</code>
Input parameter{in}	
<code>receive_message</code>	CAN message receive stuct, the structure members can refer to members of the structure Table 3-54. Structure can receive message struct
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* CAN0 FIFO0 receive message */

can_message_receive(CAN0, CAN_FIFO0, &receive_message);
```

can_fifo_release

The description of can_fifo_release is shown as below:

Table 3-69. Function can_fifo_release

Function name	can_fifo_release
Function prototype	void can_fifo_release(uint32_t can_periph, uint8_t fifo_number);
Function descriptions	release FIFO0
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
Input parameter{in}	
fifo_number	Fifo number
CAN_FIFOx	CAN_FIFOx(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-
Function name	can_fifo_release
Function prototype	void can_fifo_release(uint32_t can_periph, uint8_t fifo_number);
Function descriptions	release FIFO0
Precondition	-
The called functions	-
Input parameter{in}	

can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
fifo_number	Fifo number
CAN_FIFOx	CAN_FIFOx(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CAN0 release FIFO0*/
```

```
can_fifo_release (CAN0, CAN_FIFO0);
```

can_receive_message_length_get

The description of can_receive_message_length_get is shown as below:

Table 3-70. Function can_receive_message_length_get

Function name	can_receive_message_length_get
Function prototype	uint8_t can_receive_message_length_get(uint32_t can_periph, uint8_t fifo_number);
Function descriptions	CAN receive message length
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
fifo_number	Fifo number
CAN_FIFOx	CAN_FIFOx(x=0,1)
Output parameter{out}	

-	-
Return value	
uint8_t	0..3
Function name	can_receive_message_length_get
Function prototype	uint8_t can_receive_message_length_get(uint32_t can_periph, uint8_t fifo_number);
Function descriptions	CAN receive message length
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
fifo_number	Fifo number
CAN_FIFOx	CAN_FIFOx(x=0,1)
Output parameter{out}	
-	-
Return value	
uint8_t	0..3

Example:

```
/* CAN0 FIFO0 receive message length */

uint8_t frame_number = 0;

frame_number = can_receive_message_length_get (CAN0, CAN_FIFO0);
```

can_working_mode_set

The description of can_working_mode_set is shown as below:

Table 3-71. Function can_working_mode_set

Function name	can_working_mode_set
Function prototype	ErrStatus can_working_mode_set(uint32_t can_periph, uint8_t working_mode);

Function descriptions	set CAN working mode
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
can_working_mode	Mode select
CAN_MODE_INITIALIZE	Initialize mode
CAN_MODE_NORMAL	Normal mode
CAN_MODE_SLEEP	Sleep mode
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR

Example:

```
/* set CAN0 working at initialize mode */
can_working_mode_set (CAN0, CAN_MODE_INITIALIZE);
```

can_wakeup

The description of can_wakeup is shown as below:

Table 3-72. Function can_wakeup

Function name	can_wakeup
Function prototype	ErrStatus can_wakeup(uint32_t can_periph);
Function descriptions	wake up CAN
Precondition	-
The called functions	-
Input parameter{in}	

can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR
Function name	can_wakeup
Function prototype	ErrStatus can_wakeup(uint32_t can_periph);
Function descriptions	wake up CAN
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR

Example:

```
/* wake up CAN0 */
can_wakeup (CAN0);
```

can_error_get

The description of can_error_get is shown as below:

Table 3-73. Function can_error_get

Function name	can_error_get
Function prototype	can_error_enum can_error_get(uint32_t can_periph);
Function descriptions	get CAN error type
Precondition	-

The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
can_error_enum	0..7
Function name	can_error_get
Function prototype	can_error_enum can_error_get(uint32_t can_periph);
Function descriptions	get CAN error type
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
can_error_enum	0..7

Example:

```
/* get CAN0 error type */

can_error_get (CAN0);
```

can_receive_error_number_get

The description of can_receive_error_number_get is shown as below:

Table 3-74. Function can_receive_error_number_get

Function name	can_receive_error_number_get
Function prototype	uint8_t can_receive_error_number_get(uint32_t can_periph);

Function descriptions	get CAN receive error number
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
uint8_t	0..255
Function name	can_receive_error_number_get
Function prototype	uint8_t can_receive_error_number_get(uint32_t can_periph);
Function descriptions	get CAN receive error number
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
uint8_t	0..255

Example:

```
/* get CAN0 receive error number */

can_receive_error_number_get (CAN0);
```

can_transmit_error_number_get it

The description of can_transmit_error_number_get is shown as below:

Table 3-75. Function can_transmit_error_number_get

Function name	can_transmit_error_number_get
Function prototype	uint8_t can_transmit_error_number_get(uint32_t can_periph);
Function descriptions	get CAN transmit error number
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
uint8_t	0..255
Function name	can_transmit_error_number_get
Function prototype	uint8_t can_transmit_error_number_get(uint32_t can_periph);
Function descriptions	get CAN transmit error number
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
uint8_t	0..255

Example:

```
/* get CAN0 transmit error number */
can_transmit_error_number_get (CAN0);
```

can_flag_get

The description of can_flag_get is shown as below:

Table 3-76. Function can_flag_get

Function name	can_flag_get
Function prototype	FlagStatus can_flag_get(uint32_t can_periph, can_flag_enum flag);
Function descriptions	get CAN flag state
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0, 1)	CAN peripheral selection
Input parameter{in}	
flag	CAN flags
CAN_FLAG_MTE2	mailbox 2 transmit error
CAN_FLAG_MTE1	mailbox 1 transmit error
CAN_FLAG_MTE0	mailbox 0 transmit error
CAN_FLAG_MTF2	mailbox 2 transmit finished
CAN_FLAG_MTF1	mailbox 1 transmit finished
CAN_FLAG_MTF0	mailbox 0 transmit finished
CAN_FLAG_RFO0	receive FIFO0 overfull
CAN_FLAG_RFF0	receive FIFO0 full
CAN_FLAG_RFO1	receive FIFO1 overfull
CAN_FLAG_RFF1	receive FIFO1 full
CAN_FLAG_BOERR	bus-off error
CAN_FLAG_PERR	passive error
CAN_FLAG_WERR	warning error
Output parameter{out}	
-	-

Return value	
FlagStatus	SET / RESET

Example:

```
/* get CAN0 mailbox 0 transmit finished flag */
can_flag_get (CAN0, CAN_FLAG_MTF0);
```

can_flag_clear

The description of can_flag_clear is shown as below:

Table 3-77. Function can_flag_clear

Function name	can_flag_clear
Function prototype	void can_flag_clear(uint32_t can_periph, can_flag_enum flag);
Function descriptions	clear CAN flag state
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
flag	CAN flags
CAN_FLAG_MTE2	mailbox 2 transmit error
CAN_FLAG_MTE1	mailbox 1 transmit error
CAN_FLAG_MTE0	mailbox 0 transmit error
CAN_FLAG_MTF2	mailbox 2 transmit finished
CAN_FLAG_MTF1	mailbox 1 transmit finished
CAN_FLAG_MTF0	mailbox 0 transmit finished
CAN_FLAG_RFO0	receive FIFO0 overfull
CAN_FLAG_RFF0	receive FIFO0 full
CAN_FLAG_RFO1	receive FIFO1 overfull
CAN_FLAG_RFF1	receive FIFO1 full

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear CAN0 mailbox 0 transmit error flag*/
can_flag_clear (CAN0, CAN_FLAG_MTE0);
```

can_interrupt_enable

The description of can_interrupt_enable is shown as below:

Table 3-78. Function can_interrupt_enable

Function name	can_interrupt_enable
Function prototype	void can_interrupt_enable(uint32_t can_periph, uint32_t interrupt);
Function descriptions	enable CAN interrupt
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
interrupt	Interrupt type
CAN_INT_TME	transmit mailbox empty interrupt enable
CAN_INT_RFNE0	receive FIFO0 not empty interrupt enable
CAN_INT_RFF0	receive FIFO0 full interrupt enable
CAN_INT_RF00	receive FIFO0 overfull interrupt enable
CAN_INT_RFNE1	receive FIFO1 not empty interrupt enable
CAN_INT_RFF1	receive FIFO1 full interrupt enable
CAN_INT_RF01	receive FIFO1 overfull interrupt enable
CAN_INT_WERR	warning error interrupt enable

<i>CAN_INT_PERR</i>	passive error interrupt enable
<i>CAN_INT_BO</i>	bus-off interrupt enable
<i>CAN_INT_ERRN</i>	error number interrupt enable
<i>CAN_INT_ERR</i>	error interrupt enable
<i>CAN_INT_WU</i>	wakeup interrupt enable
<i>CAN_INT_SLPW</i>	sleep working interrupt enable
Output parameter{out}	
-	-
Return value	
-	-
Function name	can_interrupt_enable
Function prototype	void can_interrupt_enable(uint32_t can_periph, uint32_t interrupt);
Function descriptions	enable CAN interrupt
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Input parameter{in}	
interrupt	Interrupt type
<i>CAN_INT_TME</i>	transmit mailbox empty interrupt enable
<i>CAN_INT_RFNE0</i>	receive FIFO0 not empty interrupt enable
<i>CAN_INT_RFF0</i>	receive FIFO0 full interrupt enable
<i>CAN_INT_RFO0</i>	receive FIFO0 overfull interrupt enable
<i>CAN_INT_RFNE1</i>	receive FIFO1 not empty interrupt enable
<i>CAN_INT_RFF1</i>	receive FIFO1 full interrupt enable
<i>CAN_INT_RFO1</i>	receive FIFO1 overfull interrupt enable
<i>CAN_INT_WERR</i>	warning error interrupt enable

<i>CAN_INT_PERR</i>	passive error interrupt enable
<i>CAN_INT_BO</i>	bus-off interrupt enable
<i>CAN_INT_ERRN</i>	error number interrupt enable
<i>CAN_INT_ERR</i>	error interrupt enable
<i>CAN_INT_WU</i>	wakeup interrupt enable
<i>CAN_INT_SLPW</i>	sleep working interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CAN0 transmit mailbox empty interrupt */
can_interrupt_enable (CAN0, CAN_INT_TME);
```

can_interrupt_disable

The description of can_interrupt_disable is shown as below:

Table 3-79. Function can_interrupt_disable

Function name	can_interrupt_disable
Function prototype	void can_interrupt_disable(uint32_t can_periph, uint32_t interrupt);
Function descriptions	disable CAN interrupt
Precondition	-
The called functions	-
Input parameter{in}	
<i>can_periph</i>	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Input parameter{in}	
<i>interrupt</i>	Interrupt type
<i>CAN_INT_TME</i>	transmit mailbox empty interrupt enable
<i>CAN_INT_RFNE0</i>	receive FIFO0 not empty interrupt enable

<code>CAN_INT_RFF0</code>	receive FIFO0 full interrupt enable
<code>CAN_INT_RF00</code>	receive FIFO0 overfull interrupt enable
<code>CAN_INT_RFNE1</code>	receive FIFO1 not empty interrupt enable
<code>CAN_INT_RFF1</code>	receive FIFO1 full interrupt enable
<code>CAN_INT_RFO1</code>	receive FIFO1 overfull interrupt enable
<code>CAN_INT_WERR</code>	warning error interrupt enable
<code>CAN_INT_PERR</code>	passive error interrupt enable
<code>CAN_INT_BO</code>	bus-off interrupt enable
<code>CAN_INT_ERRN</code>	error number interrupt enable
<code>CAN_INT_ERR</code>	error interrupt enable
<code>CAN_INT_WU</code>	wakeup interrupt enable
<code>CAN_INT_SLPW</code>	sleep working interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CAN0 transmit mailbox empty interrupt disable */
can_interrupt_disable (CAN0, CAN_INT_TME);
```

can_interrupt_flag_get

The description of can_interrupt_flag_get is shown as below:

Table 3-80. Function can_interrupt_flag_get

Function name	<code>can_interrupt_flag_get</code>
Function prototype	<code>FlagStatus can_interrupt_flag_get(uint32_t can_periph, can_interrupt_flag_enum flag);</code>
Function descriptions	get CAN interrupt flag state
Precondition	-
The called functions	-

Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
flag	CAN interrupt flags
CAN_INT_FLAG_SLPI_F	status change interrupt flag of sleep working mode entering
CAN_INT_FLAG_WUIF	status change interrupt flag of wakeup from sleep working mode
CAN_INT_FLAG_ERRI_F	error interrupt flag
CAN_INT_FLAG_MTF2	mailbox 2 transmit finished interrupt flag
CAN_INT_FLAG_MTF1	mailbox 1 transmit finished interrupt flag
CAN_INT_FLAG_MTF0	mailbox 0 transmit finished interrupt flag
CAN_INT_FLAG_RF00	receive FIFO0 overfull interrupt flag
CAN_INT_FLAG_RF01	receive FIFO0 full interrupt flag
CAN_INT_FLAG_RF11	receive FIFO1 overfull interrupt flag
CAN_INT_FLAG_RF10	receive FIFO1 full interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* get CAN0 mailbox 0 transmit finished interrupt flag */
can_interrupt_flag_get (CAN0, CAN_INT_FLAG_MTF0);
```

can_interrupt_flag_clear

The description of can_interrupt_flag_clear is shown as below:

Table 3-81. Function can_interrupt_flag_clear

Function name	can_interrupt_flag_clear
Function prototype	void can_interrupt_flag_clear(uint32_t can_periph, can_interrupt_flag_enum

	flag);
Function descriptions	clear CAN interrupt flag state
Precondition	-
The called functions	-
Input parameter{in}	
<code>can_periph</code>	CAN peripheral
<code>CANx(x=0, 1)</code>	CAN peripheral selection
Input parameter{in}	
<code>flag</code>	CAN interrupt flags
<code>CAN_INT_FLAG_SLPIF</code>	status change interrupt flag of sleep working mode entering
<code>CAN_INT_FLAG_WUIF</code>	status change interrupt flag of wakeup from sleep working mode
<code>CAN_INT_FLAG_ERRIF</code>	error interrupt flag
<code>CAN_INT_FLAG_MTF2</code>	mailbox 2 transmit finished interrupt flag
<code>CAN_INT_FLAG_MTF1</code>	mailbox 1 transmit finished interrupt flag
<code>CAN_INT_FLAG_MTF0</code>	mailbox 0 transmit finished interrupt flag
<code>CAN_INT_FLAG_RFO0</code>	receive FIFO0 overfull interrupt flag
<code>CAN_INT_FLAG_RFF0</code>	receive FIFO0 full interrupt flag
<code>CAN_INT_FLAG_RFO1</code>	receive FIFO1 overfull interrupt flag
<code>CAN_INT_FLAG_RFF1</code>	receive FIFO1 full interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear CAN0 mailbox 0 transmit finished interrupt flag */
can_interrupt_flag_clear (CAN0, CAN_INT_FLAG_MTF0);
```

3.4. CRC

A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. The CRC registers are listed in chapter [3.4.1](#), the CRC firmware functions are introduced in chapter [3.4.2](#).

3.4.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

Table 3-82. CRC Registers

Registers	Descriptions
CRC_DATA	CRC data register
CRC_FDATA	CRC free data register
CRC_CTL	CRC control register

3.4.2. Descriptions of Peripheral functions

CRC firmware functions are listed in the table shown as below:

Table 3-83. CRC firmware function

Function name	Function description
crc_deinit	deinit CRC calculation unit
crc_data_register_reset	reset data register(CRC_DATA) to the value of 0xFFFFFFFF
crc_data_register_read	read the value of the data register
crc_free_data_register_read	read the value of the free data register
crc_free_data_register_write	write data to the free data register
crc_single_data_calculate	calculate the CRC value of a 32-bit data
crc_block_data_calculate	calculate the CRC value of an array of 32-bit values

crc_deinit

The description of crc_deinit is shown as below:

Table 3-84. Function crc_deinit

Function name	crc_deinit
Function prototype	void crc_deinit(void);
Function descriptions	deinit CRC calculation unit
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* reset crc */
crc_deinit();
```

crc_data_register_reset

The description of `crc_data_register_reset` is shown as below:

Table 3-85. Function `crc_data_register_reset`

Function name	<code>crc_data_register_reset</code>
Function prototype	<code>void crc_data_register_reset(void);</code>
Function descriptions	reset data register(CRC_DATA) to the value of 0xFFFFFFFF
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

crc_data_register_read

The description of `crc_data_register_read` is shown as below:

Table 3-86. Function `crc_data_register_read`

Function name	<code>crc_data_register_read</code>
Function prototype	<code>uint32_t crc_data_register_read(void);</code>
Function descriptions	read the value of the data register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

uint32_t	32-bit value of the data register (0-0xFFFFFFFF)
-----------------	--

Example:

```
/* read crc data register */

uint32_t crc_value = 0;

crc_value = crc_data_register_read();
```

crc_free_data_register_read

The description of `crc_free_data_register_read` is shown as below:

Table 3-87. Function `crc_free_data_register_read`

Function name	crc_free_data_register_read
Function prototype	uint8_t crc_free_data_register_read(void);
Function descriptions	read the value of the free data register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint8_t	8-bit value of the free data register (0-0xFF)

Example:

```
/* read crc free data register */

uint8_t crc_value = 0;

crc_value = crc_free_data_register_read();
```

crc_free_data_register_write

The description of `crc_free_data_register_write` is shown as below:

Table 3-88. Function `crc_free_data_register_write`

Function name	crc_free_data_register_write
Function prototype	void crc_free_data_register_write(uint8_t free_data);
Function descriptions	write data to the free data register
Precondition	-
The called functions	-
Input parameter{in}	
free_data	specified 8-bit data
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* write the free data register */

crc_free_data_register_write(0x11);
```

crc_single_data_calculate

The description of **crc_single_data_calculate** is shown as below:

Table 3-89. Function `crc_single_data_calculate`

Function name	crc_single_data_calculate
Function prototype	uint32_t crc_single_data_calculate(uint32_t sdata);
Function descriptions	calculate the CRC value of a 32-bit data
Precondition	-
The called functions	-
Input parameter{in}	
sdata	specified 32-bit data
Output parameter{out}	
-	-
Return value	
uint32_t	32-bit value calculated by CRC (0-0xFFFFFFFF)

Example:

```
/* CRC calculate a 32-bit data */

uint32_t val = 0, valcrc = 0;

val = (uint32_t)0xabcd1234;

rcu_periph_clock_enable(RCU_CRC);

valcrc = crc_single_data_calculate(val);
```

crc_block_data_calculate

The description of **crc_block_data_calculate** is shown as below:

Table 3-90. Function `crc_block_data_calculate`

Function name	crc_block_data_calculate
Function prototype	uint32_t crc_block_data_calculate(uint32_t array[], uint32_t size);
Function descriptions	calculate the CRC value of an array of 32-bit values
Precondition	-
The called functions	-

Input parameter{in}	
array	pointer to an array of 32-bit values
Input parameter{in}	
size	size of the array
Output parameter{out}	
-	-
Return value	
uint32_t	32-bit value calculated by CRC (0-0xFFFFFFFF)

Example:

```

/* CRC calculate a 32-bit data array */

#define BUFFER_SIZE      6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {
    0x00001111, 0x00002222, 0x00003333, 0x00004444, 0x00005555, 0x00006666};

rcu_periph_clock_enable(RCU_CRC);

valcrc = crc_block_data_calculate((uint32_t *) data_buffer, BUFFER_SIZE);

```

3.5. CTC

The CTC unit trims the frequency of the IRC48M which is based on an external accurate reference signal source. It can adjust the calibration value to provide a precise IRC48M clock automatically or manually. The CTC registers are listed in chapter [3.5.1](#), the CTC firmware functions are introduced in chapter [3.5.2](#)

3.5.1. Descriptions of Peripheral registers

CTC registers are listed in the table shown as below:

Table 3-91. CTC Registers

Registers	Descriptions
CTC_CTL0	CTC control register 0
CTC_CTL1	CTC control register 1
CTC_STAT	CTC status register
CTC_INTC	CTC Interrupt clear register

3.5.2. Descriptions of Peripheral functions

CTC registers are listed in the table shown as below:

Table 3-92. CTC firmware function

Function name	Function description
ctc_deinit	reset CTC clock trim controller
ctc_counter_enable	enable CTC trim counter
ctc_counter_disable	disable CTC trim counter
ctc_irc48m_trim_value_config	configure the IRC48M trim value
ctc_software_refsource_pulse_generate	generate software reference source sync pulse
ctc_hardware_trim_mode_config	configure hardware automatically trim mode
ctc_refsource_polarity_config	configure reference signal source polarity
ctc_refsource_signal_select	select reference signal source
ctc_refsource_prescaler_config	configure reference signal source prescaler
ctc_clock_limit_value_config	configure clock trim base limit value
ctc_counter_reload_value_config	configure CTC counter reload value
ctc_counter_capture_value_read	read CTC counter capture value when reference sync pulse occurred
ctc_counter_direction_read	read CTC trim counter direction when reference sync pulse occurred
ctc_counter_reload_value_read	read CTC counter reload value
ctc_irc48m_trim_value_read	read the IRC48M trim value
ctc_interrupt_enable	enable the CTC interrupt
ctc_interrupt_disable	disable the CTC interrupt
ctc_interrupt_flag_get	get CTC interrupt flag
ctc_interrupt_flag_clear	clear CTC interrupt flag
ctc_flag_get	get CTC flag
ctc_flag_clear	clear CTC flag

ctc_deinit

The description of `ctc_deinit` is shown as below:

Table 3-93. Function `ctc_deinit`

Function name	ctc_deinit
Function prototype	void <code>ctc_deinit</code> (void)
Function descriptions	Reset CTC peripheral
Precondition	-
The called functions	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset CTC */
ctc_deinit();
```

ctc_counter_enable

The description of `ctc_counter_enable` is shown as below:

Table 3-94. Function `ctc_counter_enable`

Function name	ctc_counter_enable
Function prototype	void <code>ctc_counter_enable</code> (void);
Function descriptions	enable CTC counter
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable CTC trim counter*/
ctc_counter_enable();
```

ctc_counter_disable

The description of `ctc_counter_disable` is shown as below:

Table 3-95. Function `ctc_counter_disable`

Function name	ctc_counter_disable
Function prototype	void ctc_counter_disable (void);
Function descriptions	disable CTC counter
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable CTC trim counter */
ctc_counter_disable();
```

ctc_irc48m_trim_value_config

The description of `ctc_irc48m_trim_value_config` is shown as below:

Table 3-96. Function `ctc_irc48m_trim_value_config`

Function name	ctc_irc48m_trim_value_config
Function prototype	void ctc_irc48m_trim_value_config(uint8_t trim_value);

Function descriptions	configure the IRC48M trim value
Precondition	-
The called functions	-
Input parameter{in}	
trim_value	0~63
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* IRC48M trim value configuration */
ctc_irc48m_trim_value_config (0x01);
```

ctc_software_refsource_pulse_generate

The description of `ctc_software_refsource_pulse_generate` is shown as below:

Table 3-97. Function `ctc_software_refsource_pulse_generate`

Function name	ctc_software_refsource_pulse_generate
Function prototype	void ctc_software_refsource_pulse_generate (void)
Function descriptions	generate software reference source sync pulse
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* generate reference source sync pulse */
```

`ctc_software_refsource_pulse_generate ();`

ctc_hardware_trim_mode_config

The description of `ctc_hardware_trim_mode_config` is shown as below:

Table 3-98. Function `ctc_hardware_trim_mode_config`

Function name	<code>ctc_hardware_trim_mode_config</code>
Function prototype	<code>void ctc_hardware_trim_mode_config(uint32_t hardmode);</code>
Function descriptions	configure hardware automatically trim mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>hardmode</code>	hardware automatically trim mode enable or disable
<code>CTC_HARDWARE_TRIM_MODE_ENABLE</code>	hardware automatically trim mode enable
<code>CTC_HARDWARE_TRIM_MODE_DISABLE</code>	hardware automatically trim mode disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable CTC hardware trim */

ctc.hardware.trim.mode_config(CTC_HARDWARE_TRIM_MODE_ENABLE);
```

ctc_refsource_polarity_config

The description of `ctc_refsource_polarity_config` is shown as below:

Table 3-99. Function `ctc_refsource_polarity_config`

Function name	<code>ctc_refsource_polarity_config</code>
Function prototype	<code>void ctc_refsource_polarity_config(uint32_t polarity);</code>
Function descriptions	configure reference signal source polarity
Precondition	-

The called functions		-
Input parameter{in}		
polarity		reference signal source polarity
<i>CTC_REFRESOURCE_POLARITY_FALLING</i>		reference signal source polarity is falling edge
<i>CTC_REFRESOURCE_POLARITY_RISING</i>		reference signal source polarity is rising edge
Output parameter{out}		
-		-
Return value		
-		-

Example:

```
/* set reference source polarity */
ctc_refresource_polarity_config (CTC_REFRESOURCE_POLARITY_RISING);
```

ctc_refresource_signal_select

The description of `ctc_refresource_signal_select` is shown as below:

Table 3-100. Function `ctc_refresource_signal_select`

Function name	ctc_refresource_signal_select
Function prototype	void ctc_refresource_signal_select(uint32_t refs);
Function descriptions	select reference signal source
Precondition	-
The called functions	-
Input parameter{in}	
refs	reference signal source
<i>CTC_REFRESOURCE_GPIO</i>	GPIO is selected
<i>CTC_REFRESOURCE_LXTAL</i>	LXTAL is selected
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* reference signal selection */
ctc_refsource_signal_select (CTC_REFRESOURCE_LXTAL);
```

ctc_refsource_prescaler_config

The description of `ctc_refsource_prescaler_config` is shown as below:

Table 3-101. Function `ctc_refsource_prescaler_config`

Function name	ctc_refsource_prescaler_config
Function prototype	void ctc_refsource_prescaler_config(uint32_t prescaler);
Function descriptions	configure reference signal source prescaler
Precondition	-
The called functions	-
Input parameter{in}	
prescaler	Prescaler factor
<code>CTC_REFRESOURCE_P_SC_OFF</code>	reference signal not divided
<code>CTC_REFRESOURCE_P_SC_DIV2</code>	reference signal divided by 2
<code>CTC_REFRESOURCE_P_SC_DIV4</code>	reference signal divided by 4
<code>CTC_REFRESOURCE_P_SC_DIV8</code>	reference signal divided by 8
<code>CTC_REFRESOURCE_P_SC_DIV16</code>	reference signal divided by 16
<code>CTC_REFRESOURCE_P_SC_DIV32</code>	reference signal divided by 32
<code>CTC_REFRESOURCE_P_SC_DIV64</code>	reference signal divided by 64

<i>CTC_REFRESOURCE_P</i>	reference signal divided by 128
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure reference signal source prescaler */
ctc_refresource_prescaler_config(CTC_REFRESOURCE_PSC_DIV2);
```

ctc_clock_limit_value_config

The description of **ctc_clock_limit_value_config** is shown as below:

Table 3-102. Function *ctc_clock_limit_value_config*

Function name	ctc_clock_limit_value_config
Function prototype	void ctc_clock_limit_value_config(uint8_t limit_value);
Function descriptions	configure clock trim base limit value
Precondition	-
The called functions	-
Input parameter{in}	
limit_value	0x00 - 0xFF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure clock trim base limit value */
ctc_clock_limit_value_config (0x1F);
```

ctc_counter_reload_value_config

The description of **ctc_counter_reload_value_config** is shown as below:

Table 3-103. Function ctc_counter_reload_value_config

Function name	ctc_counter_reload_value_config
Function prototype	void ctc_counter_reload_value_config(uint16_t reload_value);
Function descriptions	configure CTC counter reload value
Precondition	-
The called functions	-
Input parameter{in}	
reload_value	0x0000 - 0xFFFF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CTC counter reload value */
ctc_counter_reload_value_config (0x00FF);
```

ctc_counter_capture_value_read

The description of **ctc_counter_capture_value_read** is shown as below:

Table 3-104. Function ctc_counter_capture_value_read

Function name	ctc_counter_capture_value_read
Function prototype	uint16_t ctc_counter_capture_value_read(void);
Function descriptions	read CTC counter capture value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

uint16_t	读取计数器捕获值(0x0000 - 0xFFFF)
-----------------	---------------------------

Example:

```
/* read CTC counter capture value */
uint16_t ctc_value = 0;
ctc_value = ctc_counter_capture_value_read();
```

ctc_counter_direction_read

The description of **ctc_counter_direction_read** is shown as below:

Table 3-105. Function **ctc_counter_direction_read**

Function name	ctc_counter_direction_read
Function prototype	FlagStatus ctc_counter_direction_read(void);
Function descriptions	read CTC trim counter direction
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET(向下计数) / RESET(向上计数)

Example:

```
/* read ctc counter direction */
FlagStatus ctc_direction = SET;
ctc_direction = ctc_counter_direction_read();
```

ctc_counter_reload_value_read

The description of **ctc_counter_reload_value_read** is shown as below:

Table 3-106. Function **ctc_counter_reload_value_read**

Function name	ctc_counter_reload_value_read
Function prototype	uint16_t ctc_counter_reload_value_read(void);

Function descriptions	read CTC counter reload value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	读取计数器重载值的16位数据 (0x0000 - 0xFFFF)

Example:

```
/* read CTC counter reload value */

uint16_t ctc_reload_value = 0;

ctc_reload_value = ctc_counter_reload_value_read();
```

ctc_irc48m_trim_value_read

The description of `ctc_irc48m_trim_value_read` is shown as below:

Table 3-107. Function `ctc_irc48m_trim_value_read`

Function name	ctc_irc48m_trim_value_read
Function prototype	uint8_t ctc_irc48m_trim_value_read(void);
Function descriptions	read the IRC48M trim value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint8_t	6位IRC48M校准值 (0-63)

Example:

```

/* read the IRC48M trim value */

uint8_t ctc_trim_value = 0;

ctc_trim_value = ctc_irc48m_trim_value_read ();

```

ctc_interrupt_enable

The description of `ctc_interrupt_enable` is shown as below:

Table 3-108. Function `ctc_interrupt_enable`

Function name	ctc_interrupt_enable
Function prototype	void ctc_interrupt_enable(uint32_t interrupt);
Function descriptions	enable the CTC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	CTC interrupt
<i>CTC_INT_CKOK</i>	clock trim OK interrupt
<i>CTC_INT_CKWARN</i>	clock trim warning interrupt
<i>CTC_INT_ERR</i>	error interrupt
<i>CTC_INT_EREF</i>	expect reference interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable CTC clock trim OK interrupt */

ctc_interrupt_enable (CTC_INT_CKOK);

```

ctc_interrupt_disable

The description of `ctc_interrupt_disable` is shown as below:

Table 3-109. Function `ctc_interrupt_disable`

Function name	ctc_interrupt_disable
----------------------	-----------------------

Function prototype	void ctc_interrupt_disable(uint32_t interrupt);
Function descriptions	disable the CTC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	CTC interrupt
<i>CTC_INT_CKOK</i>	clock trim OK interrupt
<i>CTC_INT_CKWARN</i>	clock trim warning interrupt
<i>CTC_INT_ERR</i>	error interrupt
<i>CTC_INT_EREF</i>	expect reference interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable CTC clock trim OK interrupt */
ctc_interrupt_disable (CTC_INT_CKOK);
```

ctc_interrupt_flag_get

The description of `ctc_interrupt_flag_get` is shown as below:

Table 3-110. Function `ctc_interrupt_flag_get`

Function name	ctc_interrupt_flag_get
Function prototype	FlagStatus ctc_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get CTC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	CTC interrupt flag
<i>CTC_INT_FLAG_CKO</i>	clock trim OK interrupt

<i>K</i>	
<i>CTC_INT_FLAG_CKWA RN</i>	clock trim warning interrupt
<i>CTC_INT_FLAG_ERR</i>	error interrupt
<i>CTC_INT_FLAG_EREF</i>	expect reference interrupt
<i>CTC_INT_FLAG_CKE RR</i>	clock trim error bit interrupt
<i>CTC_INT_FLAG_REF MISS</i>	reference sync pulse miss interrupt
<i>CTC_INT_FLAG_TRIM ERR</i>	trim value error interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get CTC interrupt flag status */
FlagStatus state = ctc_interrupt_flag_get (CTC_INT_FLAG_CKOK);
```

ctc_interrupt_flag_clear

The description of `ctc_interrupt_flag_clear` is shown as below:

Table 3-111. Function `ctc_interrupt_flag_clear`

Function name	ctc_interrupt_flag_clear
Function prototype	void ctc_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear CTC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	CTC interrupt flag
<i>CTC_INT_FLAG_CKO K</i>	clock trim OK interrupt

<i>CTC_INT_FLAG_CKWRN</i>	clock trim warning interrupt
<i>CTC_INT_FLAG_ERR</i>	error interrupt
<i>CTC_INT_FLAG_EREF</i>	expect reference interrupt
<i>CTC_INT_FLAG_CKEERR</i>	clock trim error bit interrupt
<i>CTC_INT_FLAG_REFMISS</i>	reference sync pulse miss interrupt
<i>CTC_INT_FLAG_TRIMERR</i>	trim value error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*clear CTC interrupt flag status */
ctc_interrupt_flag_clear (CTC_INT_FLAG_CKOK);
```

ctc_flag_get

The description of **ctc_flag_get** is shown as below:

Table 3-112. Function ctc_flag_get

Function name	ctc_flag_get
Function prototype	FlagStatus ctc_flag_get(uint32_t flag);
Function descriptions	get CTC status flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	CTC status flag
<i>CTC_FLAG_CKOK</i>	clock trim OK interrupt flag
<i>CTC_FLAG_CKWARN</i>	clock trim warning interrupt flag

<i>CTC_FLAG_ERR</i>	error interrupt flag
<i>CTC_FLAG_EREF</i>	expect reference interrupt flag
<i>CTC_FLAG_CKERR</i>	clock trim error bit
<i>CTC_FLAG_REFMISS</i>	reference sync pulse miss flag
<i>CTC_FLAG_TRIMERR</i>	trim value error flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get CTC flag status */
FlagStatus state = ctc_flag_get (CTC_FLAG_CKOK);
```

ctc_flag_clear

The description of `ctc_flag_clear` is shown as below:

Table 3-113. Function `ctc_flag_clear`

Function name	ctc_flag_clear
Function prototype	void ctc_flag_clear (uint32_t flag);
Function descriptions	clear CTC status flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	CTC status flag
<i>CTC_FLAG_CKOK</i>	clock trim OK interrupt flag
<i>CTC_FLAG_CKWARN</i>	clock trim warning interrupt flag
<i>CTC_FLAG_ERR</i>	error interrupt flag
<i>CTC_FLAG_EREF</i>	expect reference interrupt flag
<i>CTC_FLAG_CKERR</i>	clock trim error bit
<i>CTC_FLAG_REFMISS</i>	reference sync pulse miss flag

<code>CTC_FLAG_TRIMERR</code>	trim value error flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear CTC flag status */

ctc_flag_clear (CTC_FLAG_CKOK);
```

3.6. DAC

The Digital-to-analog converter converts 12-bit digital data to a voltage on the external pins. The DAC registers are listed in chapter [3.6.1](#), the DAC firmware functions are introduced in chapter [3.6.2](#).

3.6.1. Descriptions of Peripheral registers

DAC registers are listed in the table shown as below:

Table 3-114. DAC Registers

Registers	Descriptions
<code>DAC_CTL</code>	DAC control register
<code>DAC_SWT</code>	DAC software trigger register
<code>DAC0_R12DH</code>	DAC0 12-bit right-aligned data holding register
<code>DAC0_L12DH</code>	DAC0 12-bit left-aligned data holding register
<code>DAC0_R8DH</code>	DAC0 8-bit right-aligned data holding register
<code>DAC1_R12DH</code>	DAC1 12-bit right-aligned data holding register
<code>DAC1_L12DH</code>	DAC1 12-bit left-aligned data holding register
<code>DAC1_R8DH</code>	DAC1 8-bit right-aligned data holding register
<code>DACC_R12DH</code>	DAC concurrent mode 12-bit right-aligned data holding register
<code>DACC_L12DH</code>	DAC concurrent mode 12-bit left-aligned data holding register
<code>DACC_R8DH</code>	DAC concurrent mode 8-bit right-aligned data holding register
<code>DAC0_DO</code>	DAC0 data output register
<code>DAC1_DO</code>	DAC1 data output register
<code>DAC_STAT</code>	DAC status register

3.6.2. Descriptions of Peripheral functions

DAC registers are listed in the table shown as below:

Table 3-115. DAC firmware function

Function name	Function description
dac_deinit	deinitialize DAC
dac_enable	enable DAC
dac_disable	disable DAC
dac_dma_enable	enable DAC DMA
dac_dma_disable	disable DAC DMA
dac_output_buffer_enable	enable DAC output buffer
dac_output_buffer_disable	disable DAC output buffer
dac_output_value_get	get the last data output value
dac_data_set	set DAC data holding register value
dac_trigger_enable	enable DAC trigger
dac_trigger_disable	disable DAC trigger
dac_trigger_source_config	configure DAC trigger source
dac_software_trigger_enable	enable DAC software trigger
dac_software_trigger_disable	disable DAC software trigger
dac_wave_mode_config	configure DAC wave mode
dac_wave_bit_width_config	configure DAC wave bit width
dac_lfsr_noise_config	configure DAC LFSR noise mode
dac_triangle_noise_config	configure DAC triangle noise mode
dac_concurrent_enable	enable DAC concurrent mode
dac_concurrent_disable	disable DAC concurrent mode
dac_concurrent_software_trigger_enable	enable DAC concurrent software trigger
dac_concurrent_software_trigger_disable	disable DAC concurrent software trigger
dac_concurrent_output_buffer_enable	enable DAC concurrent buffer function
dac_concurrent_output_buffer_disable	disable DAC concurrent buffer function
dac_concurrent_data_set	set DAC concurrent mode data holding register value
dac_concurrent_interrupt_enable	enable DAC concurrent interrupt
dac_concurrent_interrupt_disable	disable DAC concurrent interrupt
dac_flag_get	get the specified DAC flag (DAC DMA underrun flag)
dac_flag_clear	clear the specified DAC flag (DAC DMA underrun flag)
dac_interrupt_enable	enable DAC interrupt (DAC DMA underrun interrupt)
dac_interrupt_disable	disable DAC interrupt (DAC DMA underrun interrupt)
dac_interrupt_flag_get	get the specified DAC interrupt flag (DAC DMA underrun interrupt flag)
dac_interrupt_flag_clear	clear the specified DAC interrupt flag (DAC DMA underrun interrupt flag)

dac_deinit

The description of `dac_deinit` is shown as below:

Table 3-116. Function `dac_deinit`

Function name	dac_deinit
Function prototype	void dac_deinit(void);
Function descriptions	Reset DAC peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize DAC */
dac_deinit();
```

dac_enable

The description of `dac_enable` is shown as below:

Table 3-117. Function `dac_enable`

Function name	dac_enable
Function prototype	void dac_enable(uint32_t dac_periph);
Function descriptions	enable DAC
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
DACx	peripheral selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC0 */
dac_enable(DAC0);
```

dac_disable

The description of `dac_disable` is shown as below:

Table 3-118. Function `dac_disable`

Function name	dac_disable
Function prototype	void dac_disable(uint32_t dac_periph);
Function descriptions	disable DAC
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0,1)</i>	peripheral selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC0 */
dac_disable(DAC0);
```

dac_dma_enable

The description of `dac_dma_enable` is shown as below:

Table 3-119. Function `dac_dma_enable`

Function name	dac_dma_enable
Function prototype	void dac_dma_enable(uint32_t dac_periph);
Function descriptions	enable DAC DMA function
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0,1)</i>	peripheral selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC0 DMA function */
dac_dma_enable(DAC0);
```

dac_dma_disable

The description of `dac_dma_disable` is shown as below:

Table 3-120. Function `dac_dma_disable`

Function name	dac_dma_disable
Function prototype	void dac_dma_disable(uint32_t dac_periph);
Function descriptions	disable DAC DMA function
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0,1)</i>	peripheral selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC0 DMA function */

dac_dma_disable(DAC0);
```

dac_output_buffer_enable

The description of `dac_output_buffer_enable` is shown as below:

Table 3-121. Function `dac_output_buffer_enable`

Function name	dac_output_buffer_enable
Function prototype	void dac_output_buffer_enable(uint32_t dac_periph);
Function descriptions	enable DAC output buffer
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0,1)</i>	peripheral selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC0 output buffer */

dac_output_buffer_enable(DAC0);
```

dac_output_buffer_disable

The description of `dac_output_buffer_disable` is shown as below:

Table 3-122. Function `dac_output_buffer_disable`

Function name	dac_output_buffer_disable
Function prototype	void dac_output_buffer_disable(uint32_t dac_periph);
Function descriptions	disable DAC output buffer
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0,1)</i>	peripheral selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC0 output buffer */

dac_output_buffer_disable(DAC0);
```

dac_output_value_get

The description of `dac_output_value_get` is shown as below:

Table 3-123. Function `dac_output_value_get`

Function name	dac_output_value_get
Function prototype	uint16_t dac_output_value_get(uint32_t dac_periph);
Function descriptions	get DAC output value
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0,1)</i>	peripheral selection(x =0,1)
Output parameter{out}	
-	-
Return value	
uint16_t	DAC output data (0x0000 – 0x07FF)-

Example:

```
/* get DAC0 output value */

data = dac_output_value_get(DAC0);
```

dac_data_set

The description of **dac_data_set** is shown as below:

Table 3-124. Function dac_data_set

Function name	dac_data_set
Function prototype	void dac_data_set(uint32_t dac_periph, uint32_t dac_align, uint16_t data);
Function descriptions	set the DAC specified data holding register value
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0,1)</i>	peripheral selection(x =0,1)
Input parameter{in}	
dac_align	data alignment
<i>DAC_ALIGN_8B_R</i>	data right 8b alignment
<i>DAC_ALIGN_12B_R</i>	data right 12b alignment
<i>DAC_ALIGN_12B_L</i>	data left 12b alignment
Input parameter{in}	
data	data to be loaded
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the DAC0 specified data holding register value */
dac_data_set(DAC0, DAC_ALIGN_8B_R,0xff);
```

dac_trigger_enable

The description of **dac_trigger_enable** is shown as below:

Table 3-125. Function dac_trigger_enable

Function name	dac_trigger_enable
Function prototype	void dac_trigger_enable(uint32_t dac_periph);
Function descriptions	enable DAC trigger
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0,1)</i>	peripheral selection(x = 0,1)
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable DAC0 trigger */

dac_trigger_enable(DAC0);
```

dac_trigger_disable

The description of `dac_trigger_disable` is shown as below:

Table 3-126. Function `dac_trigger_disable`

Function name	dac_trigger_disable
Function prototype	void dac_trigger_disable(uint32_t dac_periph);
Function descriptions	disable DAC trigger
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0,1)</i>	peripheral selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC0 trigger */

dac_trigger_disable(DAC0);
```

dac_trigger_source_config

The description of `dac_trigger_source_config` is shown as below:

Table 3-127. Function `dac_trigger_source_config`

Function name	dac_trigger_source_config
Function prototype	void dac_trigger_source_config(uint32_t dac_periph,uint32_t triggersource);
Function descriptions	set DAC trigger source
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0,1)</i>	peripheral selection(x =0,1)

Input parameter{in}	
triggersource	external triggers of DAC
<i>DAC_TRIGGER_T1_TRGO</i>	TIMER1 TRGO
<i>DAC_TRIGGER_T3_TRGO</i>	TIMER3 TRGO
<i>DAC_TRIGGER_T4_TRGO</i>	TIMER4 TRGO
<i>DAC_TRIGGER_T5_TRGO</i>	TIMER5 TRGO
<i>DAC_TRIGGER_T6_TRGO</i>	TIMER6 TRGO
<i>DAC_TRIGGER_T7_TRGO</i>	TIMER7 TRGO
<i>DAC_TRIGGER EXTI_9</i>	EXTI interrupt line9 event
<i>DAC_TRIGGER_SOFTWARE</i>	software trigger
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* set DAC0 trigger source */
dac_trigger_source_config(DAC0, DAC_TRIGGER_T1_TRGO);

dac_software_trigger_enable
```

The description of **dac_software_trigger_enable** is shown as below:

Table 3-128. Function dac_software_trigger_enable

Function name	dac_software_trigger_enable
Function prototype	void dac_software_trigger_enable(uint32_t dac_periph);
Function descriptions	enable DAC software trigger
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0,1)</i>	peripheral selection(x =0,1)
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable DAC software trigger */

dac_software_trigger_enable(DAC0);
```

dac_software_trigger_disable

The description of `dac_software_trigger_disable` is shown as below:

Table 3-129. Function `dac_software_trigger_disable`

Function name	dac_software_trigger_disable
Function prototype	void dac_software_trigger_disable(uint32_t dac_periph);
Function descriptions	disable DAC software trigger
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0, 1)</i>	peripheral selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC0 software trigger */

dac_software_trigger_disable(DAC0);
```

dac_wave_mode_config

The description of `dac_wave_mode_config` is shown as below:

Table 3-130. Function `dac_wave_mode_config`

Function name	dac_wave_mode_config
Function prototype	void dac_wave_mode_config(uint32_t dac_periph, uint32_t wave_mode);
Function descriptions	configure DAC wave mode
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0, 1)</i>	peripheral selection(x =0,1)
Input parameter{in}	
wave_mode	wave_mode

<i>DAC_WAVE_DISABLE</i>	wave disable
<i>DAC_WAVE_MODE_LFSR</i>	LFSR noise mode
<i>DAC_WAVE_MODE_TRIANGLE</i>	triangle noise mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DAC0 wave mode */
dac_wave_mode_config(DAC0, DAC_WAVE_DISABLE);
```

dac_wave_bit_width_config

The description of **dac_wave_bit_width_config** is shown as below:

Table 3-131. Function dac_wave_bit_width_config

Function name	dac_wave_bit_width_config
Function prototype	void dac_wave_bit_width_config(uint32_t dac_periph, uint32_t bit_width);
Function descriptions	configure DAC wave bit width
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
DACx (x = 0,1)	peripheral selection(x =0,1)
Input parameter{in}	
bit_width	noise wave bit width
DAC_WAVE_BIT_WIDTH_x	noise wave bit width (x = 1..12)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DAC0 wave bit width */
dac_wave_bit_width_config(DAC0, DAC_WAVE_BIT_WIDTH_1);
```

dac_lfsr_noise_config

The description of **dac_lfsr_noise_config** is shown as below:

Table 3-132. Function dac_lfsr_noise_config

Function name	dac_lfsr_noise_config
Function prototype	void dac_lfsr_noise_config(uint32_t dac_periph, uint32_t unmask_bits);
Function descriptions	configure DAC LFSR noise mode
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0, 1)</i>	peripheral selection(x =0,1)
Input parameter{in}	
unmask_bits	unmask LFSR bits in DAC LFSR noise mode
<i>DAC_LFSR_BIT0</i>	unmask the LFSR bit 0
<i>DAC_LFSR_BITSx_0</i>	unmask the LFSR bits[x:0]
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DAC0 LFSR noise mode */

dac_lfsr_noise_config(DAC0, DAC_LFSR_BIT0);
```

dac_triangle_noise_config

The description of **dac_triangle_noise_config** is shown as below:

Table 3-133. Function dac_triangle_noise_config

Function name	dac_triangle_noise_config
Function prototype	void dac_triangle_noise_config(uint32_t dac_periph, uint32_t amplitude);
Function descriptions	configure DAC triangle noise mode
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0, 1)</i>	peripheral selection(x =0,1)
Input parameter{in}	
amplitude	triangle amplitude in DAC triangle noise mode
<i>DAC_TRIANGLE_AMPLITUDE_x</i>	triangle amplitude is x = 2 ⁿ -1 (n = 1..12)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure DAC0 triangle noise mode */

dac_triangle_noise_config(DAC0, DAC_TRIANGLE_AMPLITUDE_1);

```

dac_concurrent_enable

The description of **dac_concurrent_enable** is shown as below:

Table 3-134. Function dac_concurrent_enable

Function name	dac_concurrent_enable
Function prototype	void dac_concurrent_enable (void);
Function descriptions	enable DAC concurrent mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable DAC concurrent mode */

dac_concurrent_enable();

```

dac_concurrent_disable

The description of **dac_concurrent_disable** is shown as below:

Table 3-135. Function dac_concurrent_disable

Function name	dac_concurrent_disable
Function prototype	void dac_concurrent_disable (void);
Function descriptions	disable DAC concurrent mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC concurrent mode */
```

```
dac_concurrent_disable();
```

dac_concurrent_software_trigger_enable

The description of `dac_concurrent_software_trigger_enable` is shown as below:

Table 3-136. Function `dac_concurrent_software_trigger_enable`

Function name	dac_concurrent_software_trigger_enable
Function prototype	void dac_concurrent_software_trigger_enable (void);
Function descriptions	enable DAC concurrent software trigger function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC concurrent software trigger function */
```

```
dac_concurrent_software_trigger_enable();
```

dac_concurrent_software_trigger_disable

The description of `dac_concurrent_software_trigger_disable` is shown as below:

Table 3-137. Function `dac_concurrent_software_trigger_disable`

Function name	dac_concurrent_software_trigger_disable
Function prototype	void dac_concurrent_software_trigger_disable (void);
Function descriptions	disable DAC concurrent software trigger function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC concurrent software trigger function */
```

```
dac_concurrent_software_trigger_disable();
```

dac_concurrent_output_buffer_enable

The description of **dac_concurrent_output_buffer_enable** is shown as below:

Table 3-138. Function dac_concurrent_output_buffer_enable

Function name	dac_concurrent_output_buffer_enable
Function prototype	void dac_concurrent_output_buffer_enable(void);
Function descriptions	enable DAC concurrent buffer function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC concurrent buffer function */

dac_concurrent_output_buffer_enable();
```

dac_concurrent_output_buffer_disable

The description of **dac_concurrent_output_buffer_disable** is shown as below:

Table 3-139. Function dac_concurrent_output_buffer_disable

Function name	dac_concurrent_output_buffer_disable
Function prototype	void dac_concurrent_output_buffer_disable(void);
Function descriptions	disable DAC concurrent buffer function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC concurrent buffer function */

dac_concurrent_output_buffer_disable();
```

dac_concurrent_data_set

The description of `dac_concurrent_data_set` is shown as below:

Table 3-140. Function `dac_concurrent_data_set`

Function name	<code>dac_concurrent_data_set</code>
Function prototype	<code>void dac_concurrent_data_set(uint32_t dac_align, uint16_t data0, uint16_t data1)</code>
Function descriptions	set DAC concurrent mode data holding register value
Precondition	-
The called functions	-
Input parameter{in}	
<code>dac_align</code>	data alignment
<code>DAC_ALIGN_8B_R</code>	data right 8 bit alignment
<code>DAC_ALIGN_12B_R</code>	data right 12 bit alignment
<code>DAC_ALIGN_12B_L</code>	data left 12 bit alignment
Input parameter{in}	
<code>data0</code>	The data sending to holding register of DAC0
Input parameter{in}	
<code>data1</code>	The data sending to holding register of DAC1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set DAC concurrent mode data holding register value */

dac_concurrent_data_set(DAC_ALIGN_8B_R, 0xff, 0xff);
```

dac_concurrent_interrupt_enable

The description of `dac_concurrent_interrupt_enable` is shown as below:

Table 3-141. Function `dac_concurrent_interrupt_enable`

Function name	<code>dac_concurrent_interrupt_enable</code>
Function prototype	<code>void dac_concurrent_interrupt_enable(void);</code>
Function descriptions	enable DAC concurrent interrupt
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable DAC concurrent interrupt function */
```

```
dac_concurrent_interrupt_enable();
```

dac_concurrent_interrupt_disable

The description of `dac_concurrent_interrupt_disable` is shown as below:

Table 3-142. Function `dac_concurrent_interrupt_disable`

Function name	dac_concurrent_interrupt_disable
Function prototype	void dac_concurrent_interrupt_disable(void);
Function descriptions	disable DAC concurrent interrupt
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC concurrent interrupt function */
```

```
dac_concurrent_interrupt_disable();
```

dac_flag_get

The description of `dac_flag_get` is shown as below:

Table 3-143. Function `dac_flag_get`

Function name	dac_flag_get
Function prototype	FlagStatus dac_flag_get(uint32_t dac_periph);
Function descriptions	get the specified DAC flag (DAC DMA underrun flag)
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
DACx (x = 0,1)	peripheral selection (x =0,1)
Output parameter{out}	
-	-
Return value	

FlagStatus	SET or RESET
-------------------	--------------

Example:

```
/* get the DAC0 flag bit */

FlagStatus flag_value;

flag_value = dac_flag_get(DAC0);
```

dac_flag_clear

The description of **dac_flag_clear** is shown as below:

Table 3-144. Function dac_flag_clear

Function name	dac_flag_clear
Function prototype	void dac_flag_clear(uint32_t dac_periph);
Function descriptions	clear the specified DAC flag (DAC DMA underrun flag)
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0,1)</i>	peripheral selection (x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the DAC0 flag bit */

dac_flag_clear(DAC0);
```

dac_interrupt_enable

The description of **dac_interrupt_enable** is shown as below:

Table 3-145. Function dac_interrupt_enable

Function name	dac_interrupt_enable
Function prototype	void dac_interrupt_enable(uint32_t dac_periph);
Function descriptions	enable DAC interrupt (DAC DMA underrun interrupt)
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0,1)</i>	peripheral selection(x =0,1)
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable DAC0 interrupt */

dac_interrupt_enable(DAC0);
```

dac_interrupt_disable

函数dac_interrupt_disable描述见下表：

The description of dac_interrupt_disable is shown as below:

Table 3-146. Function dac_interrupt_disable

Function name	dac_interrupt_disable
Function prototype	void dac_interrupt_disable(uint32_t dac_periph);
Function descriptions	disable DAC interrupt (DAC DMA underrun interrupt)
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
DACx (x = 0,1)	peripheral selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC0 interrupt */

dac_interrupt_disable(DAC0);
```

dac_interrupt_flag_get

The description of dac_interrupt_flag_get is shown as below:

Table 3-147. Function dac_interrupt_flag_get

Function name	dac_interrupt_flag_get
Function prototype	FlagStatus dac_interrupt_flag_get(uint32_t dac_periph);
Function descriptions	get the specified DAC interrupt flag (DAC DMA underrun interrupt flag)
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral

<i>DACx (x = 0,1)</i>	peripheral selection (x =0,1)
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the DAC0 interrupt flag bit */

FlagStatus flag_value;

flag_value = dac_interrupt_flag_get(DAC0);

dac_interrupt_flag_clear
```

The description of `dac_interrupt_flag_clear` is shown as below:

Table 3-148. Function `dac_interrupt_flag_get`

Function name	dac_interrupt_flag_clear
Function prototype	void dac_interrupt_flag_clear(uint32_t dac_periph);
Function descriptions	clear the specified DAC interrupt flag (DAC DMA underrun interrupt flag)
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	peripheral
<i>DACx (x = 0,1)</i>	peripheral selection (x =0,1)
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* clear the DAC0 interrupt flag bit */

dac_interrupt_flag_clear(DAC0);
```

3.7. **DBG**

The DBG hold unit helps debugger to debug power saving mode. The DBG registers are listed in chapter [3.7.1](#), the DBG firmware functions are introduced in chapter [3.7.2](#).

3.7.1. **Descriptions of Peripheral registers**

DBG registers are listed in the table shown as below:

Table 3-149. DBG Registers

Registers	Descriptions
DBG_ID	DBG ID code register
DBG_CTL	DBG control register

3.7.2. Descriptions of Peripheral functions

DBG firmware functions are listed in the table shown as below:

Table 3-150. DBG firmware function

Function name	Function description
dbg_id_get	read DBG_ID code register
dbg_low_power_enable	enable low power behavior when the MCU is in debug mode
dbg_low_power_disable	disable low power behavior when the MCU is in debug mode
dbg_periph_enable	enable peripheral behavior when the MCU is in debug mode
dbg_periph_disable	disable peripheral behavior when the MCU is in debug mode
dbg_trace_pin_enable	enable trace pin assignment
dbg_trace_pin_disable	disable trace pin assignment

Enum dbg_periph_enum

Table 3-151. Enum dbg_periph_enum

Member name	Function description
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted
DBG_TIMER0_HOLD	hold TIMER0 counter when core is halted
DBG_TIMER1_HOLD	hold TIMER1 counter when core is halted
DBG_TIMER2_HOLD	hold TIMER2 counter when core is halted
DBG_TIMER3_HOLD	hold TIMER3 counter when core is halted
DBG_CAN0_HOLD	debug CAN0 kept when core is halted
DBG_I2C0_HOLD	hold I2C0 smbus when core is halted
DBG_I2C1_HOLD	hold I2C1 smbus when core is halted
DBG_TIMER4_HOLD	hold TIMER4 counter when core is halted

DBG_TIMER5_HOLD	hold TIMER5 counter when core is halted
DBG_TIMER6_HOLD	hold TIMER6 counter when core is halted
DBG_TIMER7_HOLD	hold TIMER7 counter when core is halted
DBG_CAN1_HOLD	debug CAN1 kept when core is halted
DBG_TIMER11_HOLD	hold TIMER11 counter when core is halted
DBG_TIMER12_HOLD	hold TIMER12 counter when core is halted
DBG_TIMER13_HOLD	hold TIMER13 counter when core is halted
DBG_TIMER8_HOLD	hold TIMER8 counter when core is halted
DBG_TIMER9_HOLD	hold TIMER9 counter when core is halted
DBG_TIMER10_HOLD	hold TIMER10 counter when core is halted

dbg_id_get

The description of dbg_id_get is shown as below:

Table 3-152. Function dbg_id_get

Function name	dbg_id_get
Function prototype	uint32_t dbg_id_get(void);
Function descriptions	Read DBG_ID code register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	DBG_ID code (0-0xFFFFFFFF)

Example:

```

/* read DBG_ID code register */

uint32_t id_value = 0;

id_value = dbg_id_get();

```

dbg_low_power_enable

The description of `dbg_low_power_enable` is shown as below:

Table 3-153. Function `dbg_low_power_enable`

Function name	dbg_low_power_enable
Function prototype	<code>void dbg_low_power_enable(uint32_t dbg_low_power);</code>
Function descriptions	Enable low power behavior when the mcu is in debug mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>dbg_low_power</code>	low power mode
<code>DBG_LOW_POWER_SLEEP</code>	keep debugger connection during sleep mode
<code>DBG_LOW_POWER_DEEPSLEEP</code>	keep debugger connection during deepsleep mode
<code>DBG_LOW_POWER_STANDBY</code>	keep debugger connection during standby mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable low power behavior when the mcu is in debug mode */
dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

dbg_low_power_disable

The description of `dbg_low_power_disable` is shown as below:

Table 3-154. Function `dbg_low_power_disable`

Function name	dbg_low_power_disable
Function prototype	<code>void dbg_low_power_disable(uint32_t dbg_low_power);</code>
Function descriptions	Disable low power behavior when the mcu is in debug mode

Precondition	-
The called functions	-
Input parameter{in}	
dbg_low_power	low power mode
DBG_LOW_POWER_SLEEP	keep debugger connection during sleep mode
DBG_LOW_POWER_DEEPSLEEP	keep debugger connection during deepsleep mode
DBG_LOW_POWER_STANDBY	keep debugger connection during standby mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable low power behavior when the mcu is in debug mode */

dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

dbg_periph_enable

The description of `dbg_periph_enable` is shown as below:

Table 3-155. Function `dbg_periph_enable`

Function name	dbg_periph_enable
Function prototype	void dbg_periph_enable(dbg_periph_enum dbg_periph);
Function descriptions	Enable peripheral behavior when the mcu is in debug mode
Precondition	-
The called functions	-
Input parameter{in}	
dbg_periph	Peripheral refer to Table 3-151. Enum <code>dbg_periph_enum</code>
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted

<i>DBG_CANx_HOLD</i>	x=0,1, hold CANx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1, hold I2Cx smbus when core is halted
<i>DBG_TIMERx_HOLD</i>	x=0,1,2,3,4,5,6,7,8,9,10,11,12,13, hold TIMERx counter when core is halted
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable peripheral behavior when the mcu is in debug mode */
dbg_periph_enable(DBG_TIMER0_HOLD);
```

dbg_periph_disable

The description of `dbg_periph_disable` is shown as below:

Table 3-156. Function `dbg_periph_disable`

Function name	dbg_periph_disable
Function prototype	void dbg_periph_disable(dbg_periph_enum dbg_periph);
Function descriptions	Disable peripheral behavior when the mcu is in debug mode
Precondition	-
The called functions	-
Input parameter{in}	
<i>dbg_periph</i>	peripheral refer to Table 3-151. Enum <code>dbg_periph_enum</code>
<i>DBG_FWDGT_HOLD</i>	debug FWDGT kept when core is halted
<i>DBG_WWDGT_HOLD</i>	debug WWDGT kept when core is halted
<i>DBG_CANx_HOLD</i>	x=0,1, hold CANx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1, hold I2Cx smbus when core is halted
<i>DBG_TIMERx_HOLD</i>	x=0,1,2,3,4,5,6,7,8,9,10,11,12,13, hold TIMERx counter when core is halted
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable peripheral behavior when the mcu is in debug mode */

dbg_periph_disable(DBG_TIMER0_HOLD);
```

dbg_trace_pin_enable

The description of dbg_trace_pin_enable is shown as below:

Table 3-157. Function dbg_trace_pin_enable

Function name	dbg_trace_pin_enable
Function prototype	void dbg_trace_pin_enable(void);
Function descriptions	Enable trace pin assignment
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable trace pin assignment */

dbg_trace_pin_enable();
```

dbg_trace_pin_disable

The description of dbg_trace_pin_disable is shown as below:

Table 3-158. Function dbg_trace_pin_disable

Function name	dbg_trace_pin_disable
Function prototype	void dbg_trace_pin_disable(void);
Function descriptions	Disable trace pin assignment
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable trace pin assignment */
dbg_trace_pin_disable();
```

3.8. DCI

The DCI is a parallel interface able to capture video or picture from a camera via Digital Camera Interface. The DCI registers are listed in chapter [3.8.1](#). the DCI firmware functions are introduced in chapter [3.8.2](#).

3.8.1. Descriptions of Peripheral registers

DCI registers are listed in the table shown as below:

Table 3-159. DCI Registers

Registers	Descriptions
DCI_CTL	DCI control register
DCI_STAT0	DCI status register 0
DCI_STAT1	DCI status register 1
DCI_INTEN	DCI interrupt enable register
DCI_INTF	DCI interrupt flag register
DCI_INTC	DCI interrupt clear register
DCI_SC	DCI synchronization codes register
DCI_SCUMSK	DCI synchronization codes unmask register
DCI_CWSPOS	DCI cropping window start position register

Registers	Descriptions
DCI_CWSZ	DCI cropping window size register
DCI_DATA	DCI data register

3.8.2. Descriptions of Peripheral functions

DCI firmware functions are listed in the table shown as below:

Table 3-160. DCI firmware function

Function name	Function description
dci_deinit	DCI deinit
dci_init	initialize DCI registers
dci_enable	enable DCI function
dci_disable	disble DCI function
dci_capture_enable	enable DCI capture
dci_capture_disable	disble DCI capture
dci_jpeg_enable	enable DCI jpeg mode
dci_jpeg_disable	disble DCI jpeg mode
dci_crop_window_enable	enable cropping window function
dci_crop_window_disable	disble cropping window function
dci_crop_window_config	config DCI cropping window
dci_embedded_sync_enable	enable embedded synchronous mode
dci_embedded_sync_disable	disble embedded synchronous mode
dci_sync_codes_config	config synchronous codes in embedded synchronous mode
dci_sync_codes_unmask_config	config synchronous codes unmask in embedded synchronous mode
dci_data_read	read DCI data register
dci_flag_get	get specified flag
dci_interrupt_enable	enable specified DCI interrupt
dci_interrupt_disable	disble specified DCI interrupt
dci_interrupt_flag_get	get specified interrupt flag

Function name	Function description
dci_interrupt_flag_clear	clear specified interrupt flag

Structure dci_parameter_struct

Table 3-161. Structure dci_parameter_struct

Member name	Function description
capture_mode	DCI capture mode: DCI_CAPTURE_MODE_CONTINUOUS / DCI_CAPTURE_MODE_SNAPSHOT
clock_polarity	clock polarity selection: DCI_CK_POLARITY_FALLING / DCI_CK_POLARITY_RISING
hsync_polarity	horizontal polarity selection: DCI_HSYNC_POLARITY_LOW / DCI_HSYNC_POLARITY_HIGH
vsync_polarity	vertical polarity selection: DCI_VSYNC_POLARITY_LOW / DCI_VSYNC_POLARITY_HIGH
frame_rate	frame capture rate: DCI_FRAME_RATE_ALL / DCI_FRAME_RATE_1_2 / DCI_FRAME_RATE_1_4
interface_format	digital camera interface format: DCI_INTERFACE_FORMAT_8BITS / DCI_INTERFACE_FORMAT_10BITS / DCI_INTERFACE_FORMAT_12BITS / DCI_INTERFACE_FORMAT_14BITS

dci_deinit

The description of dci_deinit is shown as below:

Table 3-162. Function dci_deinit

Function name	dci_deinit
Function prototype	void dci_deinit(void);
Function descriptions	DCI deinit
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DCI deinit */
```

```
dci_deinit ( );
```

dci_init

The description of dci_init is shown as below:

Table 3-163. Function dci_init

Function name	dci_init
Function prototype	void dci_init(dci_parameter_struct* dci_struct);
Function descriptions	initialize DCI registers
Precondition	-
The called functions	-
Input parameter{in}	
dci_struct	DCI parameter initialization stuct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize DCI registers */
```

```
dci_parameter_struct dci_struct;
```

```
dci_struct.capture_mode = DCI_CAPTURE_MODE_CONTINUOUS;
```

```
dci_struct.clock_polarity = DCI_CK_POLARITY_RISING;
```

```
dci_struct.hsync_polarity = DCI_HSYNC_POLARITY_LOW;
```

```
dci_struct.vsync_polarity = DCI_VSYNC_POLARITY_LOW;
```

```
dci_struct.frame_rate = DCI_FRAME_RATE_ALL;
dci_struct.interface_format = DCI_INTERFACE_FORMAT_8BITS;
dci_init (&dci_struct);
```

dci_enable

The description of dci_enable is shown as below:

Table 3-164. Function dci_enable

Function name	dci_enable
Function prototype	void dci_enable(void);
Function descriptions	enable DCI function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DCI function */
dci_enable( );
```

dci_disable

The description of dci_disable is shown as below:

Table 3-165. Function dci_disable

Function name	dci_disable
Function prototype	void dci_disable(void);
Function descriptions	disable DCI function
Precondition	-
The called functions	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DCI function */
dci_disable( );
```

dci_capture_enable

The description of dci_capture_enable is shown as below:

Table 3-166. Function dci_capture_enable

Function name	dci_capture_enable
Function prototype	void dci_capture_enable(void);
Function descriptions	enable DCI capture
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DCI capture */
dci_capture_enable( );
```

dci_capture_disable

The description of dci_capture_disable is shown as below:

Table 3-167. Function dci_capture_disable

Function name	dci_capture_disable
Function prototype	void dci_capture_disable(void);
Function descriptions	disable DCI capture
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DCI capture */

dci_capture_disable( );
```

dci_jpeg_enable

The description of dci_jpeg_enable is shown as below:

Table 3-168. Function dci_jpeg_enable

Function name	dci_jpeg_enable
Function prototype	void dci_jpeg_enable(void);
Function descriptions	enable DCI jpeg mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable DCI jpeg mode */
dci_jpeg_enable( );
```

dci_jpeg_disable

The description of dci_jpeg_disable is shown as below:

Table 3-169. Function dci_jpeg_disable

Function name	dci_jpeg_disable
Function prototype	void dci_jpeg_disable(void);
Function descriptions	disable DCI jpeg mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DCI jpeg mode */
dci_jpeg_disable( );
```

dci_crop_window_enable

The description of dci_crop_window_enable is shown as below:

Table 3-170. Function dci_crop_window_enable

Function name	dci_crop_window_enable
Function prototype	void dci_crop_window_enable(void);
Function descriptions	enable cropping window function
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable cropping window function */
dci_crop_window_enable( );
```

dci_crop_window_disable

The description of dci_crop_window_disable is shown as below:

Table 3-171. Function dci_crop_window_disable

Function name	dci_crop_window_disable
Function prototype	void dci_crop_window_disable(void);
Function descriptions	disable cropping window function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable cropping window function */
dci_crop_window_disable( );
```

dci_crop_window_config

The description of dci_crop_window_config is shown as below:

Table 3-172. Function dci_crop_window_config

Function name	dci_crop_window_config
Function prototype	void dci_crop_window_config(uint16_t start_x, uint16_t start_y, uint16_t size_width, uint16_t size_height);
Function descriptions	config DCI cropping window
Precondition	-
The called functions	-
Input parameter{in}	
start_x	window horizontal start position
Input parameter{in}	
start_y	window vertical start position
Input parameter{in}	
size_width	window horizontal size
Input parameter{in}	
size_height	window vertical size
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config DCI cropping window */
dci_crop_window_config (0x800, 0x600, 0x100, 0x100);
```

dci_embedded_sync_enable

The description of dci_embedded_sync_enable is shown as below:

Table 3-173. Function dci_embedded_sync_enable

Function name	dci_embedded_sync_enable
----------------------	--------------------------

Function prototype	void dci_embedded_sync_enable(void);
Function descriptions	enable embedded synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable embedded synchronous mode */
dci_embedded_sync_enable( );
```

dci_embedded_sync_disable

The description of dci_embedded_sync_disable is shown as below:

Table 3-174. Function dci_embedded_sync_disable

Function name	dci_embedded_sync_disable
Function prototype	void dci_embedded_sync_disable(void);
Function descriptions	disable embedded synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable embedded synchronous mode */
dci_embedded_sync_disable( )
```

dci_sync_codes_config

The description of dci_sync_codes_config is shown as below:

Table 3-175. Function dci_sync_codes_config

Function name	dci_sync_codes_config
Function prototype	void dci_sync_codes_config(uint8_t frame_start, uint8_t line_start, uint8_t line_end, uint8_t frame_end);
Function descriptions	config synchronous codes in embedded synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
frame_start	frame start code in embedded synchronous mode
Input parameter{in}	
line_start	line start code in embedded synchronous mode
Input parameter{in}	
line_end	line end code in embedded synchronous mode
Input parameter{in}	
frame_end	frame end code in embedded synchronous mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config synchronous codes in embedded synchronous mode */
dci_sync_codes_config (0x10, 0x10, 0x20, 0x20)
```

dci_sync_codes_unmask_config

The description of dci_sync_codes_unmask_config is shown as below:

Table 3-176. Function dci_sync_codes_unmask_config

Function name	dci_sync_codes_unmask_config
Function prototype	void dci_sync_codes_unmask_config(uint8_t frame_start, uint8_t line_start, uint8_t line_end, uint8_t frame_end);
Function descriptions	config synchronous codes unmask in embedded synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
frame_start	frame start code in embedded synchronous mode
Input parameter{in}	
line_start	line start code in embedded synchronous mode
Input parameter{in}	
line_end	line end code in embedded synchronous mode
Input parameter{in}	
frame_end	frame end code in embedded synchronous mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config synchronous codes unmask in embedded synchronous mode */
dci_sync_codes_unmask_config (0x10, 0x10, 0x20, 0x20)
```

dci_data_read

The description of dci_data_read is shown as below:

Table 3-177. Function dci_data_read

Function name	dci_data_read
Function prototype	uint32_t dci_data_read(void);
Function descriptions	read DCI data register
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	0x00 – 0xFFFFFFFF

Example:

```
/* read DCI data register */

uint32_t data = dci_data_read( );
```

dci_flag_get

The description of dci_flag_get is shown as below:

Table 3-178. Function dci_flag_get

Function name	dci_flag_get
Function prototype	FlagStatus dci_flag_get(uint32_t flag);
Function descriptions	get specified flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	DCI flag
<i>DCI_FLAG_HS</i>	HS line status
<i>DCI_FLAG_VS</i>	VS line status
<i>DCI_FLAG_FV</i>	FIFO valid
<i>DCI_FLAG_EF</i>	end of frame flag
<i>DCI_FLAG_OVR</i>	FIFO overrun flag
<i>DCI_FLAG_ESE</i>	embedded synchronous error flag
<i>DCI_FLAG_VSYNC</i>	vsync flag
<i>DCI_FLAG_EL</i>	end of line flag

Output parameter{out}	
-	-
Return value	
FlagStatus	FlagStatus: SET or RESET

Example:

```
/* get specified flag */
FlagStatus status = dci_flag_get (DCI_FLAG_HS);
```

dci_interrupt_enable

The description of **dci_interrupt_enable** is shown as below:

Table 3-179. Function dci_interrupt_enable

Function name	dci_interrupt_enable
Function prototype	void dci_interrupt_enable(uint32_t interrupt);
Function descriptions	enable specified DCI interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	DCI interrupt
<i>DCI_INT_EF</i>	end of frame interrupt
<i>DCI_INT_OVR</i>	FIFO overrun interrupt
<i>DCI_INT_ESE</i>	embedded synchronous error interrupt
<i>DCI_INT_VSYNC</i>	vsync interrupt
<i>DCI_INT_EL</i>	end of line interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable specified DCI interrupt */
```

```
dci_interrupt_enable (DCI_INT_EF);
```

dci_interrupt_disable

The description of dci_interrupt_disable is shown as below:

Table 3-180. Function dci_interrupt_disable

Function name	dci_interrupt_disable
Function prototype	void dci_interrupt_disable(uint32_t interrupt);
Function descriptions	disable specified DCI interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	DCI interrupt
<i>DCI_INT_EF</i>	end of frame interrupt
<i>DCI_INT_OVR</i>	FIFO overrun interrupt
<i>DCI_INT_ESE</i>	embedded synchronous error interrupt
<i>DCI_INT_VSYNC</i>	vsync interrupt
<i>DCI_INT_EL</i>	end of line interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable specified DCI interrupt */
dci_interrupt_disable (DCI_INT_EF);
```

dci_interrupt_flag_get

The description of dci_interrupt_flag_get is shown as below:

Table 3-181. Function dci_interrupt_flag_get

Function name	dci_interrupt_flag_get
Function prototype	FlagStatus dci_interrupt_flag_get(uint32_t interrupt);

Function descriptions	get specified interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	DCI interrupt
<i>DCI_INT_FLAG_EF</i>	end of frame interrupt flag
<i>DCI_INT_FLAG_OVR</i>	FIFO overrun interrupt flag
<i>DCI_INT_FLAG_ESE</i>	embedded synchronous error interrupt flag
<i>DCI_INT_FLAG_VSYN_C</i>	vsync interrupt flag
<i>DCI_INT_FLAG_EL</i>	end of line interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	FlagStatus: SET or RESET

Example:

```
/* get specified interrupt flag */

FlagStatus status = dci_interrupt_flag_get (DCI_INT_FLAG_EF);
```

dci_interrupt_flag_clear

The description of `dci_interrupt_flag_clear` is shown as below:

Table 3-182. Function `dci_interrupt_flag_clear`

Function name	dci_interrupt_flag_clear
Function prototype	void dci_interrupt_flag_clear(uint32_t interrupt);
Function descriptions	clear specified interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	DCI interrupt

<i>DCI_INT_FLAG_EF</i>	end of frame interrupt flag
<i>DCI_INT_FLAG_OVR</i>	FIFO overrun interrupt flag
<i>DCI_INT_FLAG_ESE</i>	embedded synchronous error interrupt flag
<i>DCI_INT_FLAG_VSYN_C</i>	vsync interrupt flag
<i>DCI_INT_FLAG_EL</i>	end of line interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*clear specified interrupt flag */
dcı_interrupt_flag_clear (DCI_INT_FLAG_EF);
```

3.9. DMA

The direct memory access (DMA) controller provides a hardware method of transferring data between peripherals and/or memory without intervention from the CPU, thereby freeing up bandwidth for other system functions. The DMA registers are listed in chapter [3.9.1](#), the DMA firmware functions are introduced in chapter [3.9.2](#).

3.9.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

Table 3-183. DMA Registers

Registers	Descriptions
DMA_INTF	Interrupt flag register
DMA_INTC	Interrupt flag clear register
DMA_CHxCTL (x=0..7)	Channel x control register
DMA_CHxCNT (x=0..7)	Channel x counter register
DMA_CHxPADDR	Channel x peripheral base address register

Registers	Descriptions
(x=0..7)	
DMA_CHxM0ADDR (x=0..7)	Channel x memory 0 base address register
DMA_CHxM1ADDR (x=0..7)	Channel x memory 1 base address register
DMA_CHxFCTL (x=0..7)	Channel x FIFO control register

3.9.2. Descriptions of Peripheral functions

DMA firmware functions are listed in the table shown as below:

Table 3-184. DMA firmware function

Function name	Function description
dma_deinit	deinitialize DMA a channel registers
dma_single_data_mode_init	DMA single data mode initialize
dma_multi_data_mode_init	DMA multi data mode initialize
dma_periph_address_config	set DMA peripheral base address
dma_memory_address_config	set DMA memory base address
dma_transfer_number_config	set the number of remaining data to be transferred by the DMA
dma_transfer_number_get	get the number of remaining data to be transferred by the DMA
dma_priority_config	configure priority level of DMA channel
dma_memory_burst_beats_config	configure transfer burst beats of memory
dma_periph_burst_beats_config	configure transfer burst beats of peripheral
dma_memory_width_config	configure transfer data size of memory
dma_periph_width_config	configure transfer data size of peripheral
dma_memory_address_generation_config	configure next address increasement algorithm of memory
dma_peripheral_address_generation_config	configure next address increasement algorithm of peripheral

Function name	Function description
dma_circulation_enable	enable DMA circulation mode
dma_circulation_disable	disable DMA circulation mode
dma_channel_enable	enable DMA channel
dma_channel_disable	disable DMA channel
dma_transfer_direction_config	configure the direction of data transfer on the channel
dma_switch_buffer_mode_config	DMA switch buffer mode config
dma_using_memory_get	DMA using memory get
dma_channel_subperipheral_select	DMA channel peripheral select
dma_flow_controller_config	DMA flow controller configure
dma_switch_buffer_mode_enable	DMA flow controller enable
dma_fifo_status_get	DMA FIFO status get
dma_flag_get	check DMA flag is set or not
dma_flag_clear	clear DMA a channel flag
dma_interrupt_flag_get	check DMA flag is set or not
dma_interrupt_flag_clear	clear DMA a channel flag
dma_interrupt_enable	enable DMA interrupt
dma_interrupt_disable	disable DMA interrupt

Structure `dma_multi_data_parameter_struct`

Table 3-185. Structure `dma_multi_data_parameter_struct`

Member name	Function description
periph_addr	peripheral base address
periph_width	transfer data size of peripheral
periph_inc	peripheral increasing mode
memory0_addr	memory 0 base address
memory_width	transfer data size of memory
memory_inc	memory increasing mode
memory_burst_widt	memory burst width

<code>h</code>	
<code>periph_burst_width</code>	periph burst width
<code>critical_value</code>	DMA circular value
<code>circular_mode</code>	DMA circular mode
<code>direction</code>	channel data transfer direction
<code>number</code>	channel transfer number
<code>priority</code>	channel priority level

Structure `dma_single_data_parameter_struct`

Table 3-186. Structure `dma_single_data_parameter_struct`

Member name	Function description
<code>periph_addr</code>	peripheral base address
<code>periph_inc</code>	peripheral increasing mode
<code>memory0_addr</code>	memory 0 base address
<code>memory_inc</code>	memory increasing mode
<code>periph_memory_width</code>	transfer data size of peripheral
<code>circular_mode</code>	DMA circular mode
<code>direction</code>	channel data transfer direction
<code>number</code>	channel transfer number
<code>priority</code>	channel priority level

`dma_deinit`

The description of `dma_deinit` is shown as below:

Table 3-187. Function `dma_deinit`

Function name	<code>dma_deinit</code>
Function prototype	<code>void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);</code>
Function descriptions	deinitialize DMA a channel registers
Precondition	-
The called functions	-

Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0,1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..7)	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA0 channel0 initialize */
dma_deinit(DMA0, DMA_CH0);
```

dma_single_data_mode_init

The description of **dma_single_data_mode_init** is shown as below:

Table 3-188. Function `dma_single_data_mode_init`

Function name	dma_single_data_mode_init
Function prototype	void dma_single_data_mode_init(uint32_t dma_periph,dma_channel_enum channelx,dma_single_data_parameter_struct* init_struct);
Function descriptions	initialize DMA single data mode
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0,1)	DMAx(x=0,1)
Input parameter{in}	
channelx	specify which DMA channel is initialized
DMA_CHx(x=0..7)	DMA_CHx(x=0..7)
Input parameter{ in }	

init_struct	a dma_single_data_parameter_struct address, the structure members can refer to Table 3-186. Structure dma_single_data_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize DMA single data mode */
dma_single_data_parameter_struct dma_single_data_struct;
dma_single_data_mode_init(DMAx0, DMA_CH0, &dma_single_data_struct);
```

dma_multi_data_mode_init

The description of `dma_multi_data_mode_init` is shown as below:

Table 3-189. Function `dma_multi_data_mode_init`

Function name	dma_multi_data_mode_init
Function prototype	void dma_multi_data_mode_init(uint32_t dma_periph,dma_channel_enum channelx,dma_multi_data_parameter_struct* init_struct);
Function descriptions	initialize DMA multi data mode
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0,1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..7)	DMA channel selection
Input parameter{in}	
init_struct	Structure for initialization, the structure members can refer to Table 3-189. Function <code>dma_multi_data_mode_init</code>
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* initialize DMA multi data mode */
dma_multi_data_parameter_struct dma_multi_data_struct;
dma_init(DMA0, DMA_CH0, &dma_multi_data_struct);
```

dma_periph_address_config

The description of **dma_periph_address_config** is shown as below:

Table 3-190. Function `dma_periph_address_config`

Function name	dma_periph_address_config
Function prototype	void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
Function descriptions	set DMA peripheral base address
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
address	peripheral base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
#define BANK0_WRITE_START_ADDR ((uint32_t)0x08004000)
```

```
dma_periph_address_config(DMA0, DMA_CH0, BANK0_WRITE_START_ADDR);
```

dma_memory_address_config

The description of `dma_memory_address_config` is shown as below:

Table 3-191. Function `dma_memory_address_config`

Function name	dma_memory_address_config
Function prototype	void dma_memory_address_config(uint32_t dma_periph,dma_channel_enum channelx,uint8_t memory_flag,uint32_t address);
Function descriptions	set DMA memory base address
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
memory_flag	DMA memory
<i>DMA_MEMORY_x(x=0, 1)</i>	DMA memory selection
Input parameter{in}	
address	memory base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
uint8_t g_destbuf[TRANSFER_NUM];
```

```
dma_memory_address_config(DMA0, DMA_CH0, DMA_MEMORY_0, (uint32_t) g_destbuf);
```

dma_transfer_number_config

The description of `dma_transfer_number_config` is shown as below:

Table 3-192. Function `dma_transfer_number_config`

Function name	dma_transfer_number_config
Function prototype	void dma_transfer_number_config(uint32_t dma_periph,dma_channel_enum channelx,uint32_t number);
Function descriptions	set the number of remaining data to be transferred by the DMA
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
number	the number of remaining data to be transferred by the DMA
<i>0-0xffff</i>	number
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
#define TRANSFER_NUM          0x400
dma_transfer_number_config(DMA0, DMA_CH0, TRANSFER_NUM);
```

dma_transfer_number_get

The description of `dma_transfer_number_get` is shown as below:

Table 3-193. Function dma_transfer_number_get

Function name	dma_transfer_number_get
Function prototype	uint32_t dma_transfer_number_get(uint32_t dma_periph,dma_channel_enum channelx);
Function descriptions	get the number of remaining data to be transferred by the DMA
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0, 1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..7)	DMA channel selection
Output parameter{out}	
-	-
Return value	
uint32_t(0-0xffff)	the number of remaining data to be transferred by the DMA(0-0xffff)

Example:

```
uint32_t number = 0;  
  
number = dma_transfer_number_get(DMA0, DMA_CH0);
```

dma_priority_config

The description of **dma_priority_config** is shown as below:

Table 3-194. Function dma_priority_config

Function name	dma_priority_config
Function prototype	void dma_priority_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priority);
Function descriptions	configure priority level of DMA channel
Precondition	-
The called functions	-

Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
priority	priority Level of this channel
<i>DMA_PRIORITY_LOW</i>	low priority
<i>DMA_PRIORITY_MEDIUM</i>	medium priority
<i>DMA_PRIORITY_HIGH</i>	high priority
<i>DMA_PRIORITY_ULTRA_HIGH</i>	ultra high priority
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_priority_config(DMA0, DMA_CH0, DMA_PRIORITY_ULTRA_HIGH);
```

dma_memory_burst_beats_config

The description of `dma_memory_burst_beats_config` is shown as below:

Table 3-195. Function `dma_memory_burst_beats_config`

Function name	dma_memory_burst_beats_config
Function prototype	void dma_memory_burst_beats_config (uint32_t dma_periph,dma_channel_enum channelx,uint32_t mbeat);
Function descriptions	configure transfer burst beats of memory
Precondition	-
The called functions	-

Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
mbeat	transfer burst beats
<i>DMA_MEMORY_BURS_T_SINGLE</i>	memory transfer single burst
<i>DMA_MEMORY_BURS_T_4_BEAT</i>	memory transfer 4-beat burst
<i>DMA_MEMORY_BURS_T_8_BEAT</i>	memory transfer 8-beat burst
<i>DMA_MEMORY_BURS_T_16_BEAT</i>	memory transfer 16-beat burst
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_memory_burst_beats_config (DMA0, DMA_CH0, DMA_MEMORY_BURST_8_BEAT);
```

dma_periph_burst_beats_config

The description of `dma_periph_burst_beats_config` is shown as below:

Table 3-196. Function `dma_periph_burst_beats_config`

Function name	dma_periph_burst_beats_config
Function prototype	void dma_periph_burst_beats_config (uint32_t dma_periph,dma_channel_enum channelx,uint32_t pbeat);
Function descriptions	configure transfer burst beats of peripheral
Precondition	-

The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
pbeat	transfer burst beats
<i>DMA_PERIPH_BURST_SINGLE</i>	peripheral transfer single burst
<i>DMA_PERIPH_BURST_4_BEAT</i>	peripheral transfer 4-beat burst
<i>DMA_PERIPH_BURST_8_BEAT</i>	peripheral transfer 8-beat burst
<i>DMA_PERIPH_BURST_16_BEAT</i>	peripheral transfer 16-beat burst
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_periph_burst_beats_config(DMA0, DMA_CH0, DMA_PERIPH_BURST_8_BEAT);
```

dma_memory_width_config

The description of `dma_memory_width_config` is shown as below:

Table 3-197. Function `dma_memory_width_config`

Function name	dma_memory_width_config
Function prototype	void dma_memory_width_config(uint32_t dma_periph,dma_channel_enum channelx,uint32_t mszie);
Function descriptions	configure transfer data size of memory

Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
msize	transfer data size of memory
<i>DMA_MEMORY_WIDTH_8BIT</i>	transfer data size of memory is 8-bit
<i>DMA_MEMORY_WIDTH_16BIT</i>	transfer data size of memory is 16-bit
<i>DMA_MEMORY_WIDTH_32BIT</i>	transfer data size of memory is 32-bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_memory_width_config(DMA0, DMA_CH0, DMA_MEMORY_WIDTH_8BIT);
```

dma_periph_width_config

The description of **dma_periph_width_config** is shown as below:

Table 3-198. Function `dma_periph_width_config`

Function name	<code>dma_periph_width_config</code>
Function prototype	<code>void dma_periph_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t pwidth);</code>
Function descriptions	configure transfer data size of peripheral

Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
psize	transfer data size of peripheral
<i>DMA_PERIPHERAL_WIDTH_8BIT</i>	transfer data size of peripheral is 8-bit
<i>DMA_PERIPHERAL_WIDTH_16BIT</i>	transfer data size of peripheral is 16-bit
<i>DMA_PERIPHERAL_WIDTH_32BIT</i>	transfer data size of peripheral is 32-bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_periph_width_config(DMA0, DMA_CH0, DMA_PERIPHERAL_WIDTH_8BIT);
```

dma_memory_address_generation_config

The description of `dma_memory_address_generation_config` is shown as below:

Table 3-199. Function `dma_memory_address_generation_config`

Function name	<code>dma_memory_address_generation_config</code>
Function prototype	<code>void dma_memory_address_generation_config(uint32_t dma_periph,dma_channel_enum channelx,uint8_t generation_algorithm);</code>
Function descriptions	configure memory address generation generation_algorithm

Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
generation_algorithm	the address generation algorithm
<i>DMA_MEMORY_INCR_EASE_ENABLE</i>	next address of memory is increasing address mode
<i>DMA_MEMORY_INCR_EASE_DISABLE</i>	next address of memory is fixed address mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_memory_address_generation_config(DMA0, DMA_CH0, DMA_MEMORY_INCREASE_ENABLE);
```

dma_peripheral_address_generation_config

The description of `dma_peripheral_address_generation_config` is shown as below:

Table 3-200. Function `dma_peripheral_address_generation_config`

Function name	dma_peripheral_address_generation_config
Function prototype	void dma_peripheral_address_generation_config(uint32_t dma_periph,dma_channel_enum channelx,uint8_t generation_algorithm);
Function descriptions	configure peripheral address generation_algorithm
Precondition	-

The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
generation_algorithm	the address generation algorithm
<i>DMA_PERIPH_INCRE ASE_ENABLE</i>	next address of peripheral is increasing address mode
<i>DMA_PERIPH_INCRE ASE_DISABLE</i>	next address of peripheral is fixed address mode
<i>DMA_PERIPH_INCRE ASE_FIX</i>	increasing steps of peripheral address is fixed
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_peripheral_address_generation_config (DMA0, DMA_CH0, DMA_PERIPH_INCRE
ASE_ENABLE);
```

dma_circulation_enable

The description of **dma_circulation_enable** is shown as below:

Table 3-201. Function `dma_circulation_enable`

Function name	dma_circulation_enable
Function prototype	void dma_circulation_enable(uint32_t dma_periph,dma_channel_enum channelx);
Function descriptions	enable DMA circulation mode
Precondition	-

The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_circulation_enable (DMA0, DMA_CH0);
```

dma_circulation_disable

The description of `dma_circulation_disable` is shown as below:

Table 3-202. Function `dma_circulation_disable`

Function name	dma_circulation_disable
Function prototype	vvoid dma_circulation_disable(uint32_t dma_periph,dma_channel_enum channelx);
Function descriptions	disable DMA circulation mode
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_circulation_disable (DMA0, DMA_CH0);
```

dma_channel_enable

The description of **dma_channel_enable** is shown as below:

Table 3-203. Function `dma_channel_enable`

Function name	dma_channel_enable
Function prototype	void dma_channel_enable(uint32_t dma_periph,dma_channel_enum channelx);
Function descriptions	enable DMA channel
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMA(x=0,1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..7)	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_channel_enable (DMA0, DMA_CH0);
```

dma_channel_disable

The description of `dma_channel_disable` is shown as below:

Table 3-204. Function `dma_channel_disable`

Function name	dma_channel_disable
Function prototype	<code>void dma_channel_disable(uint32_t dma_periph,dma_channel_enum channelx);</code>
Function descriptions	disable DMA channel
Precondition	-
The called functions	-
Input parameter{in}	
<code>dma_periph</code>	DMA peripheral
<code>DMAx(x=0,1)</code>	DMA peripheral selection
Input parameter{in}	
<code>channelx</code>	DMA channel
<code>DMA_CHx(x=0..7)</code>	DMA channel selection
Output parameter{out}	
-	-
Return value	
<code>FlagStatus</code>	SET or RESET

Example:

```
dma_channel_disable (DMA0, DMA_CH0);
```

`dma_transfer_direction_config`

The description of `dma_transfer_direction_config` is shown as below:

Table 3-205. Function `dma_transfer_direction_config`

Function name	dma_transfer_direction_config
Function prototype	<code>void dma_transfer_direction_config(uint32_t dma_periph,dma_channel_enum channelx,uint8_t direction);</code>
Function descriptions	configure the direction of data transfer on the channel
Precondition	-
The called functions	-

Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
direction	specify the direction of data transfer
<i>DMA_PERIPH_TO_MEMORY</i>	read from peripheral and write to memory
<i>DMA_MEMORY_TO_PERIPH</i>	read from memory and write to peripheral
<i>DMA_MEMORY_TO_MEMORY</i>	read from memory and write to memory
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_transfer_direction_config (DMA0, DMA_CH0, DMA_PERIPH_TO_MEMORY);
```

dma_switch_buffer_mode_config

The description of `dma_switch_buffer_mode_config` is shown as below:

Table 3-206. Function `dma_switch_buffer_mode_config`

Function name	dma_switch_buffer_mode_config
Function prototype	void dma_switch_buffer_mode_config(uint32_t dma_periph,dma_channel_enum channelx,uint32_t memory1_addr,uint32_t memory_select);
Function descriptions	DMA switch buffer mode config
Precondition	-
The called functions	-

Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
memory1_addr	memory1 base address
Input parameter{in}	
memory_select	DMA_MEMORY_0 or DMA_MEMORY_1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_switch_buffer_mode_config(DMA0, DMA_CH0,0xaabbccdd,DMA_MEMORY_0);
```

dma_using_memory_get

The description of `dma_using_memory_get` is shown as below:

Table 3-207. Function `dma_using_memory_get`

Function name	dma_using_memory_get
Function prototype	uint32_t dma_using_memory_get(uint32_t dma_periph,dma_channel_enum channelx);
Function descriptions	DMA using memory get
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection

Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
uint32_t	the using memory

Example:

```
uint32_t using_memory = dma_using_memory_get(DMA0, DMA_CH0);
```

dma_channel_subperipheral_select

The description of `dma_channel_subperipheral_select` is shown as below:

Table 3-208. Function `dma_channel_subperipheral_select`

Function name	<code>dma_channel_subperipheral_select</code>
Function prototype	<code>void dma_channel_subperipheral_select(uint32_t dma_periph,dma_channel_enum channelx,dma_subperipheral_enum sub_periph);</code>
Function descriptions	DMA channel peripheral select
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
sub_periph	specify DMA channel peripheral
<i>DMA_SUBPERIx(x=0..7)</i>	specify DMA channel peripheral select

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_channel_subperipheral_select (DMA0, DMA_CH0, DMA_SUBPERI0);
```

dma_flow_controller_config

The description of `dma_flow_controller_config` is shown as below:

Table 3-209. Function `dma_flow_controller_config`

Function name	dma_flow_controller_config
Function prototype	void dma_flow_controller_config(uint32_t dma_periph,dma_channel_enum channelx,uint32_t controller);
Function descriptions	DMA flow controller configure
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMA(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
controller	specify DMA flow controller
<i>DMA_FLOW_CONTROLLER_DMA</i>	DMA is the flow controller
<i>DMA_FLOW_CONTROLLER_PERI</i>	peripheral is the flow controller
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
dma_flow_controller_config (DMA0, DMA_CH0, DMA_FLOW_CONTROLLER_DMA);
```

dma_switch_buffer_mode_enable

The description of `dma_switch_buffer_mode_enable` is shown as below:

Table 3-210. Function `dma_switch_buffer_mode_enable`

Function name	dma_switch_buffer_mode_enable
Function prototype	void dma_switch_buffer_mode_enable(uint32_t dma_periph,dma_channel_enum channelx,ControlStatus newvalue);
Function descriptions	DMA switch buffer mode enable
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0..1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
newvalue	ENABLE or DISABLE
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_switch_buffer_mode_enable(DMA0, DMA_CH0, ENABLE);
```

dma_fifo_status_get

The description of `dma_fifo_status_get` is shown as below:

Table 3-211. Function `dma_fifo_status_get`

Function name	dma_fifo_status_get
Function prototype	<code>uint32_t dma_fifo_status_get(uint32_t dma_periph,dma_channel_enum channelx);</code>
Function descriptions	DMA FIFO status get
Precondition	-
The called functions	-
Input parameter{in}	
<code>dma_periph</code>	DMA peripheral
<code>DMAx(x=0,1)</code>	DMA peripheral selection
Input parameter{in}	
<code>channelx</code>	DMA channel
<code>DMA_CHx(x=0..7)</code>	DMA channel selection
Output parameter{out}	
-	-
Return value	
<code>uint32_t</code>	the using memory

Example:

```
uint32_t using_memory dma_fifo_status_get (DMA0, DMA_CH0);
```

dma_flag_get

The description of `dma_flag_get` is shown as below:

Table 3-212. Function `dma_flag_get`

Function name	dma_flag_get
Function prototype	<code>FlagStatus dma_flag_get(uint32_t dma_periph,dma_channel_enum channelx,uint32_t flag);</code>
Function descriptions	get DMA flag is set or not
Precondition	-

The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
flag	specify get which flag
<i>DMA_FLAG_FEE</i>	FIFO error and exception flag
<i>DMA_FLAG_SDE</i>	single data mode exception flag
<i>DMA_FLAG_TAE</i>	transfer access error flag
<i>DMA_FLAG_HTF</i>	half transfer finish flag
<i>DMA_FLAG_FTF</i>	full transfer finish flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
FlagStatus flag_status = dma_flag_get (DMA0, DMA_CH0, DMA_FLAG_FEE);
```

dma_flag_clear

The description of `dma_flag_clear` is shown as below:

Table 3-213. Function `dma_flag_clear`

Function name	<code>dma_flag_clear</code>
Function prototype	<code>void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);</code>
Function descriptions	clear DMA a channel flag
Precondition	-

The called functions		-
Input parameter{in}		
dma_periph		DMA peripheral
DMAx(x=0, 1)		DMA peripheral selection
Input parameter{in}		
channelx		DMA channel
DMA_CHx(x=0..7)		DMA channel selection
Input parameter{in}		
flag		specify get which flag
DMA_FLAG_FEE		FIFO error and exception flag
DMA_FLAG_SDE		single data mode exception flag
DMA_FLAG_TAE		transfer access error flag
DMA_FLAG_HTF		half transfer finish flag
DMA_FLAG_FTF		full transfer finish flag
Output parameter{out}		
-		-
Return value		
-		-

Example:

```
dma_flag_clear(DMA0, DMA_CH0, DMA_FLAG_FTF);
```

dma_interrupt_flag_get

The description of dma_interrupt_flag_get is shown as below:

Table 3-214. Function dma_interrupt_flag_get

Function name	dma_interrupt_flag_get
Function prototype	FlagStatus dma_interrupt_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t interrupt);
Function descriptions	check DMA flag and interrupt enable bit is set or not
Precondition	-

The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
interrupt	specify get which flag
<i>DMA_INT_FLAG_FEE</i>	FIFO error and exception flag
<i>DMA_INT_FLAG_SDE</i>	single data mode exception flag
<i>DMA_INT_FLAG_TAE</i>	transfer access error flag
<i>DMA_INT_FLAG_HTF</i>	half transfer finish flag
<i>DMA_INT_FLAG_FTF</i>	full transfer finish flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
}
```

dma_interrupt_flag_clear

The description of `dma_interrupt_flag_clear` is shown as below:

Table 3-215. Function `dma_interrupt_flag_clear`

Function name	dma_interrupt_flag_clear
Function prototype	void dma_interrupt_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t interrupt);
Function descriptions	clear DMA a channel interrupt flag

Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
interrupt	specify get which flag
<i>DMA_INT_FLAG_FEE</i>	FIFO error and exception flag
<i>DMA_INT_FLAG_SDE</i>	single data mode exception flag
<i>DMA_INT_FLAG_TAE</i>	transfer access error flag
<i>DMA_INT_FLAGHTF</i>	half transfer finish flag
<i>DMA_INT_FLAG_FTF</i>	full transger finish flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_FTF);
}
```

dma_interrupt_enable

The description of **dma_interrupt_enable** is shown as below:

Table 3-216. Function `dma_interrupt_enable`

Function name	dma_interrupt_enable
Function prototype	void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);

Function descriptions	enable DMA interrupt
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0, 1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..7)	DMA channel selection
Input parameter{in}	
source	DMA interrupt source
DMA_CHXCTL_SDEIE	single data mode exception interrupt enable
DMA_CHXCTL_TAEIE	transfer access error interrupt enable
DMA_CHXCTL_HTFIE	half transfer finish interrupt enable
DMA_CHXCTL_FTFIE	full transfer finish interrupt enable
DMA_CHXFCTL_FEEIE	FIFO exception interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA0 channel0 interrupt configuration */

dma_interrupt_enable(DMA0, DMA_CH0, DMA_CHXCTL_SDEIE);
```

dma_interrupt_disable

The description of `dma_interrupt_disable` is shown as below:

Table 3-217. Function `dma_interrupt_disable`

Function name	dma_interrupt_disable
----------------------	-----------------------

Function prototype	void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
Function descriptions	disable DMA interrupt
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
source	DMA interrupt source
<i>DMA_CHXCTL_SDEIE</i>	single data mode exception interrupt enable
<i>DMA_CHXCTL_TAEIE</i>	transfer access error interrupt enable
<i>DMA_CHXCTL_HTFIE</i>	half transfer finish interrupt enable
<i>DMA_CHXCTL_FTFIE</i>	full transfer finish interrupt enable
<i>DMA_CHXFCTL_FEEIE</i>	FIFO exception interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA0 channel0 interrupt configuration */

dma_interrupt_disable(DMA0, DMA_CH0, DMA_CHXCTL_SDEIE);
```

3.10. ENET

There is a media access controller (MAC) designed in Ethernet module to support

10/100Mbps interface speed. For more efficient data transfer between Ethernet and memory, a DMA controller is designed in this module. The support interface protocol for Ethernet is media independent interface (MII) and reduced media independent interface (RMII). The ENET registers are listed in chapter [3.10.1](#), the ENET firmware functions are introduced in chapter [3.10.2](#).

3.10.1. Descriptions of Peripheral registers

ENET registers are listed in the table shown as below:

Table 3-218. ENET Registers

Registers	Descriptions
ENET_MAC_CFG	MAC configuration register
ENET_MAC_FRMF	MAC frame filter register
ENET_MAC_HLH	MAC hash list high register
ENET_MAC_HLL	MAC hash list low register
ENET_MAC_PHY_CTL	MAC PHY control register
ENET_MAC_PHY_DATA	MAC PHY data register
ENET_MAC_FCTL	MAC flow control register
ENET_MAC_VLT	MAC VLAN tag register
ENET_MAC_RWFF	MAC remote wakeup frame filter register
ENET_MAC_WUM	MAC wakeup management register
ENET_MAC_DBG	MAC debug register
ENET_MAC_INTF	MAC interrupt flag register
ENET_MAC_INTMS_K	MAC interrupt mask register
ENET_MAC_ADDR_0H	MAC address 0 high register
ENET_MAC_ADDR_0L	MAC address 0 low register
ENET_MAC_ADDR_1H	MAC address 1 high register

Registers	Descriptions
ENET_MAC_ADDR 1L	MAC address 1 low register
ENET_MAC_ADDT 2H	MAC address 2 high register
ENET_MAC_ADDR 2L	MAC address 2 low register
ENET_MAC_ADDR 3H	MAC address 3 high register
ENET_MAC_ADDR 3L	MAC address 3 low register
ENET_MAC_FCTH	MAC flow control threshold register
ENET_MSC_CTL	MSC control register
ENET_MSC_RINTF	MSC receive interrupt flag register
ENET_MSC_TINTF	MSC transmit interrupt flag register
ENET_MSC_RINT MSK	MSC receive interrupt mask register
ENET_MSC_TINTM SK	MSC transmit interrupt mask register
ENET_MSC_SCCN T	MSC transmitted good frames after a single collision counter register
ENET_MSC_MSCC NT	MSC transmitted good frames after more than a single collision counter register
ENET_MSC_TGFC NT	MSC transmitted good frames counter register
ENET_MSC_RFCE CNT	MSC received frames with CRC error counter register
ENET_MSC_RFAE CNT	MSC received frames with alignment error counter register
ENET_MSC_RGUF CNT	MSC received good unicast frames counter register
ENET_PTP_TSCTL	PTP time stamp control register
ENET_PTP_SSINC	PTP subsecond increment register

Registers	Descriptions
ENET_PTP_TSH	PTP time stamp high register
ENET_PTP_TSL	PTP time stamp low register
ENET_PTP_TSUH	PTP time stamp update high register
ENET_PTP_TSUL	PTP time stamp update low register
ENET_PTP_TSADD END	PTP time stamp addend register
ENET_PTP_ETH	PTP expected time high register
ENET_PTP_ETL	PTP expected time low register
ENET_PTP_TSFLAG	PTP time stamp flag register
ENET_PTP_PPSCTL L	PTP PPS control register
ENET_DMA_BCTL	DMA bus control register
ENET_DMA_TPEN	DMA transmit poll enable register
ENET_DMA_RPEN	DMA receive poll enable register
ENET_DMA_RDTA DDR	DMA receive descriptor table address register
ENET_DMA_TDТА DDR	DMA transmit descriptor table address register
ENET_DMA_STAT	DMA status register
ENET_DMA_CTL	DMA control register
ENET_DMA_INTEN	DMA interrupt enable register
ENET_DMA_MFBO CNT	DMA missed frame and buffer overflow counter register
ENET_DMA_RSWD C	DMA receive state watchdog counter register
ENET_DMA_CTDA DDR	DMA current transmit descriptor address register
ENET_DMA_CRDA DDR	DMA current receive descriptor address register
ENET_DMA_CTBA	DMA current transmit buffer address register

Registers	Descriptions
DDR	
ENET_DMA_CRBA DDR	DMA current receive buffer address register

3.10.2. Descriptions of Peripheral functions

ENET firmware functions are listed in the table shown as below:

Table 3-219. ENET firmware function

Function name	Function description
main function	
enet_deinit	deinitialize the ENET, and reset structure parameters for ENET initialization
enet_initpara_config	configure the parameters which are usually less cared for initialization
enet_init	initialize ENET peripheral with generally concerned parameters and the less cared parameters
enet_software_reset	reset all core internal registers located in CLK_TX and CLK_RX
enet_rxframe_size_get	check receive frame valid and return frame size
enet_descriptors_chain_init	initialize the dma tx/rx descriptors's parameters in chain mode
enet_descriptors_ring_init	initialize the dma tx/rx descriptors's parameters in ring mode
enet_frame_receive	handle current received frame data to application buffer
enet_frame_transmit	handle application buffer data to transmit it
enet_transmit_checksum_config	configure the transmit IP frame checksum offload calculation and insertion
enet_enable	ENET Tx and Rx function enable (include MAC and DMA module)
enet_disable	ENET Tx and Rx function disable (include MAC and DMA module)
enet_mac_address_set	configure MAC address
enet_mac_address_get	get MAC address
enet_flag_get	get the ENET MAC/MSC/PTP/DMA status flag

Function name	Function description
enet_flag_clear	clear the ENET DMA status flag
enet_interrupt_enable	enable ENET MAC/MSC/DMA interrupt
enet_interrupt_disable	disable ENET MAC/MSC/DMA interrupt
enet_interrupt_flag_get	get ENET MAC/MSC/DMA interrupt flag
enet_interrupt_flag_clear	clear ENET DMA interrupt flag
MAC function	
enet_tx_enable	ENET Tx function enable (include MAC and DMA module)
enet_tx_disable	ENET Tx function disable (include MAC and DMA module)
enet_rx_enable	ENET Rx function enable (include MAC and DMA module)
enet_rx_disable	ENET Rx function disable (include MAC and DMA module)
enet_registers_get	put registers value into the application buffer
enet_debug_status_get	get the enet debug status from the debug register
enet_address_filter_enable	enable the MAC address filter
enet_address_filter_disable	disable the MAC address filter
enet_address_filter_config	configure the MAC address filter
enet_phy_config	PHY interface configuration (configure SMI clock and reset PHY chip)
enet_phy_write_read	write to/read from a PHY register
enet_phyloopback_enable	enable the loopback function of phy chip
enet_phyloopback_disable	disable the loopback function of phy chip
enet_forward_feature_enable	enable ENET forward feature
enet_forward_feature_disable	disable ENET forward feature
enet_filter_feature_enable	enable ENET filter feature
enet_filter_feature_disable	disable ENET filter feature
flow control function	
enet_pauseframe_generate	generate the pause frame, ENET will send pause frame after enable transmit flow control
enet_pauseframe_detect_config	configure the pause frame detect type

Function name	Function description
enet_pauseframe_config	configure the pause frame parameters
enet_flowcontrol_threshold_config	configure the threshold of the flow control(deactive and active threshold)
enet_flowcontrol_feature_enable	enable ENET flow control feature
enet_flowcontrol_feature_disable	disable ENET flow control feature
DMA function	
enet_dmaprocess_state_get	get the dma transmit/receive process state
enet_dmaprocess_resume	poll the dma transmission/reception enable
enet_rxprocess_check_recovery	check and recover the Rx process
enet_txfifo_flush	flush the ENET transmit fifo, and wait until the flush operation completes
enet_current_desc_address_get	get the transmit/receive address of current descriptor, or current buffer, or descriptor table
enet_desc_information_get	get the Tx or Rx descriptor information
enet_missed_frame_counter_get	get the number of missed frames during receiving
descriptor function	
enet_desc_flag_get	get the bit flag of ENET dma descriptor
enet_desc_flag_set	set the bit flag of ENET dma tx descriptor
enet_desc_flag_clear	clear the bit flag of ENET dma tx descriptor
enet_rx_desc_immediate_receive_complete_interrupt	when receiving the completed, set RS bit in ENET_DMA_STAT register will immediately set
enet_rx_desc_delay_receive_complete_interrupt	when receiving the completed, set RS bit in ENET_DMA_STAT register will is set after a configurable delay time
enet_rxframe_drop	drop current receive frame
enet_dma_feature_enable	enable DMA feature
enet_dma_feature_disable	disable DMA feature
enhanced descriptor	
enet_rx_desc_enhanced_status_get	get the bit of extended status flag in ENET DMA descriptor

Function name	Function description
enet_desc_select_enhanced_mode	configure descriptor to work in enhanced mode
enet_ptp_enhanced_descriptors_chain_init	initialize the dma Tx/Rx descriptors's parameters in enhanced chain mode with ptp function
enet_ptp_enhanced_descriptors_ring_init	initialize the dma Tx/Rx descriptors's parameters in enhanced ring mode with ptp function
enet_ptpframe_receive_enhanced_mode	receive a packet data with timestamp values to application buffer, when the DMA is in enhanced mode
enet_ptpframe_transmit_enhanced_mode	send data with timestamp values in application buffer as a transmit packet, when the DMA is in enhanced mode
normal descriptor	
enet_desc_select_normal_mode	configure descriptor to work in normal mode
enet_ptp_normal_descriptors_chain_init	initialize the dma Tx/Rx descriptors's parameters in normal chain mode with ptp function
enet_ptp_normal_descriptors_ring_init	initialize the dma Tx/Rx descriptors's parameters in normal ring mode with ptp function
enet_ptpframe_receive_normal_mode	receive a packet data with timestamp values to application buffer, when the DMA is in normal mode
enet_ptpframe_transmit_normal_mode	send data with timestamp values in application buffer as a transmit packet, when the DMA is in normal mode
WUM function	
enet_wum_filter_register_pointer_reset	wakeup frame filter register pointer reset
enet_wum_filter_config	set the remote wakeup frame registers
enet_wum_feature_enable	enable wakeup management features
enet_wum_feature_disable	disable wakeup management features
MSC function	
enet_msc_counters_reset	reset the MAC statistics counters
enet_msc_feature_enable	enable the MAC statistics counter features
enet_msc_feature_disable	disable the MAC statistics counter features
enet_msc_counters_preset_config	configure MAC statistics counters preset mode

Function name	Function description
enet_msc_counters_get	get MAC statistics counter
PTP function	
enet_ptp_feature_enable	enable the PTP features
enet_ptp_feature_disable	disable the PTP features
enet_ptp_timestamp_function_config	configure the PTP timestamp function
enet_ptp_subsecond_increment_config	configure the PTP system time subsecond increment value
enet_ptp_timestamp_addend_config	adjusting the PTP clock frequency only in fine update mode
enet_ptp_timestamp_update_config	initializing or adding/subtracting to second of the PTP system time
enet_ptp_expected_time_config	configure the PTP expected target time
enet_ptp_system_time_get	get the PTP current system time
enet_ptp_pps_output_frequency_config	configure the PPS output frequency
Other	
enet_initpara_reset	reset the ENET initpara struct, call it before using enet_initpara_config()

Structure `enet_initpara_struct`

Table 3-220. Structure `enet_initpara_struct`

member name	Function description
option_enable	select which function to configure
forward_frame	frame forward related parameters
dmabus_mode	DMA bus mode related parameters
dma_maxburst	DMA max burst related parameters
dma_arbitration	DMA Tx and Rx arbitration related parameters
store_forward_mode	store forward mode related parameters
dma_function	DMA control related parameters
vlan_config	VLAN tag related parameters

flow_control	flow control related parameters
hashtable_high	hash list high 32-bit related parameters
hashtable_low	hash list low 32-bit related parameters
framesfilter_mode	frame filter control related parameters
halfduplex_param	halfduplex related parameters
timer_config	frame timer related parameters
interframegap	inter frame gap related parameters

Structure enet_descriptors_struct

Table 3-221. Structure enet_descriptors_struct

member name	Function description
status	status
control_buffer_size	control and buffer1, buffer2 lengths
buffer1_addr	buffer1 address pointer/timestamp low
buffer2_next_desc_addr	buffer2 or next descriptor address pointer/timestamp high
extended_status	extended status
reserved	reserved
timestamp_low	timestamp low
timestamp_high	timestamp high

Structure enet_ptp_systime_struct

Table 3-222. Structure enet_ptp_systime_struct

member name	Function description
second	second of system time
nanosecond	nanosecond of system time
sign	sign of system time

enet_deinit

The description of enet_deinit is shown as below:

Table 3-223. Function enet_deinit

Function name	enet_deinit
Function prototype	void enet_deinit(void);
Function descriptions	deinitialize the ENET, and reset structure parameters for ENET initialization
Precondition	-
The called functions	rcu_periph_reset_enable() /rcu_periph_reset_disable() /enet_initpara_reset()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize the ENET */
enet_deinit( );
```

enet_initpara_config

The description of enet_initpara_config is shown as below:

Table 3-224. Function enet_initpara_config

Function name	enet_initpara_config
Function prototype	void enet_initpara_config(enet_option_enum option, uint32_t para)
Function descriptions	configure the parameters which are usually less cared for initialization, this function must be called before enet_init(), otherwise configuration will be no effect
Precondition	enet_initpara_reset(void)
The called functions	-
Input parameter{in}	
option	different function option, which is related to several parameters only one parameter can be selected which is shown as below
FORWARD_OPTION	choose to configure the frame forward related parameters

<i>DMABUS_OPTION</i>	choose to configure the DMA bus mode related parameters
<i>DMA_MAXBURST_OPTION</i>	choose to configure the DMA max burst related parameters
<i>DMA_ARBITRATION_OPTION</i>	choose to configure the DMA arbitration related parameters
<i>STORE_OPTION</i>	choose to configure the store forward mode related parameters
<i>DMA_OPTION</i>	choose to configure the DMA related parameters
<i>VLAN_OPTION</i>	choose to configure vlan related parameters
<i>FLOWCTL_OPTION</i>	choose to configure flow control related parameters
<i>HASHH_OPTION</i>	choose to configure hash high
<i>HASHL_OPTION</i>	choose to configure hash low
<i>FILTER_OPTION</i>	choose to configure frame filter related parameters
<i>HALFDUPLEX_OPTION</i>	choose to configure halfduplex mode related parameters
<i>TIMER_OPTION</i>	choose to configure time counter related parameters
<i>INTERFRAMEGAP_OPTION</i>	choose to configure the inter frame gap related parameters
Input parameter{in}	
para (the value according to the parameter option)	all the related values should be configured which are shown as below example: para = (value1 value2 value3...)
When value of parameter option is <i>FORWARD_OPTION</i>	
value1	<i>ENET_AUTO_PADCRC_DROP_ENABLE / ENET_AUTO_PADCRC_DROP_DISABLE</i>
value2	<i>ENET_FORWARD_ERRFRAMES_ENABLE / ENET_FORWARD_ERRFRAMES_DISABLE</i>
value3	<i>ENET_FORWARD_UNDERSZ_GOODFRAMES_ENABLE / ENET_FORWARD_UNDERSZ_GOODFRAMES_DISABLE</i>
value4	<i>ENET_FORWARD_UNDERSZ_GOODFRAMES_ENABLE / ENET_FORWARD_UNDERSZ_GOODFRAMES_DISABLE</i>
When value of parameter option is <i>DMABUS_OPTION</i>	
value1	<i>ENET_ADDRESS_ALIGN_ENABLE / ENET_ADDRESS_ALIGN_DISABLE</i>

value2	<i>ENET_FIXED_BURST_ENABLE / ENET_FIXED_BURST_DISABLE</i>
value3	<i>ENET_MIXED_BURST_ENABLE / ENET_MIXED_BURST_DISABLE</i>
When value of parameter option is <i>DMA_MAXBURST_OPTION</i>	
value1	<i>ENET_RXDP_1BEAT / ENET_RXDP_2BEAT / ENET_RXDP_4BEAT / ENET_RXDP_8BEAT / ENET_RXDP_16BEAT / ENET_RXDP_32BEAT / ENET_RXDP_4xPGBL_4BEAT / ENET_RXDP_4xPGBL_8BEAT / ENET_RXDP_4xPGBL_16BEAT / ENET_RXDP_4xPGBL_32BEAT / ENET_RXDP_4xPGBL_64BEAT / ENET_RXDP_4xPGBL_128BEAT</i>
value2	<i>ENET_PGBL_1BEAT / ENET_PGBL_2BEAT / ENET_PGBL_4BEAT / ENET_PGBL_8BEAT / ENET_PGBL_16BEAT / ENET_PGBL_32BEAT / ENET_PGBL_4xPGBL_4BEAT / ENET_PGBL_4xPGBL_8BEAT / ENET_PGBL_4xPGBL_16BEAT / ENET_PGBL_4xPGBL_32BEAT / ENET_PGBL_4xPGBL_64BEAT / ENET_PGBL_4xPGBL_128BEAT</i>
value3	<i>ENET_RXTX_DIFFERENT_PGBL / ENET_RXTX_SAME_PGBL</i>
When value of parameter option is <i>DMA_ARBITRATION_OPTION</i>	
value1	<i>ENET_ARBITRATION_RXPRIORTX</i>
value2	<i>ENET_ARBITRATION_RXTX_1_1 / ENET_ARBITRATION_RXTX_2_1 / ENET_ARBITRATION_RXTX_3_1 / ENET_ARBITRATION_RXTX_4_1</i>
When value of parameter option is <i>STORE_OPTION</i>	
value1	<i>ENET_RX_MODE_STOREFORWARD / ENET_RX_MODE_CUTTHROUGH</i>
value2	<i>ENET_TX_MODE_STOREFORWARD / ENET_TX_MODE_CUTTHROUGH</i>
value3	<i>ENET_RX_THRESHOLD_64BYTES / ENET_RX_THRESHOLD_32BYTES / ENET_RX_THRESHOLD_96BYTES / ENET_RX_THRESHOLD_128BYTES</i>
value4	<i>ENET_TX_THRESHOLD_64BYTES / ENET_TX_THRESHOLD_128BYTES / ENET_TX_THRESHOLD_192BYTES / ENET_TX_THRESHOLD_256BYTES / ENET_TX_THRESHOLD_40BYTES / ENET_TX_THRESHOLD_32BYTES / ENET_TX_THRESHOLD_24BYTES / ENET_TX_THRESHOLD_16BYTES</i>
When value of parameter option is <i>DMA_OPTION</i>	
value1	<i>ENET_FLUSH_RXFRAME_ENABLE / ENET_FLUSH_RXFRAME_DISABLE</i>
value2	<i>ENET_SECONDFRAME_OPT_ENABLE / ENET_SECONDFRAME_OPT_DISABLE</i>

value3	<i>ENET_ENHANCED_DESCRIPTOR/ENET_NORMAL_DESCRIPTOR</i>
When value of parameter option is <i>VLAN_OPTION</i>	
value1	<i>ENET_VLANTAGCOMPARISON_12BIT/</i> <i>ENET_VLANTAGCOMPARISON_16BIT</i>
value2	<i>MAC_VLT_VLTI(regval)</i>
When value of parameter option is <i>FLOWCTL_OPTION</i>	
value1	<i>MAC_FCTL_PTM(regval)</i>
value2	<i>ENET_ZERO_QUANTA_PAUSE_ENABLE/</i> <i>ENET_ZERO_QUANTA_PAUSE_DISABLE</i>
value3	<i>ENET_PAUSETIME_MINUS4 / ENET_PAUSETIME_MINUS28 /</i> <i>ENET_PAUSETIME_MINUS144/ENET_PAUSETIME_MINUS256</i>
value4	<i>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDTECT /</i> <i>ENET_UNIQUE_PAUSEDTECT</i>
value5	<i>ENET_RX_FLOWCONTROL_ENABLE /</i> <i>ENET_RX_FLOWCONTROL_DISABLE</i>
value6	<i>ENET_TX_FLOWCONTROL_ENABLE /</i> <i>ENET_TX_FLOWCONTROL_DISABLE</i>
When value of parameter option is <i>HASHH_OPTION</i>	
value1	0x0~0xFFFF FFFFU
When value of parameter option is <i>HASHL_OPTION</i>	
value1	0x0~0xFFFF FFFFU
When value of parameter option is <i>FILTER_OPTION</i>	
value1	<i>ENET_SRC_FILTER_NORMAL_ENABLE /</i> <i>ENET_SRC_FILTER_INVERSE_ENABLE / ENET_SRC_FILTER_DISABLE</i>
value2	<i>ENET_DEST_FILTER_INVERSE_ENABLE /</i> <i>ENET_DEST_FILTER_INVERSE_DISABLE</i>
value3	<i>ENET_MULTICAST_FILTER_HASH_OR_PERFECT /</i> <i>ENET_MULTICAST_FILTER_HASH /</i> <i>ENET_MULTICAST_FILTER_PERFECT /</i> <i>ENET_MULTICAST_FILTER_NONE</i>
value4	<i>ENET_UNICAST_FILTER_EITHER / ENET_UNICAST_FILTER_HASH /</i> <i>ENET_UNICAST_FILTER_PERFECT</i>

value5	<i>ENET_PCFRM_PREVENT_ALL / ENET_PCFRM_PREVENT_PAUSEFRAME / ENET_PCFRM_FORWARD_ALL / ENET_PCFRM_FORWARD_FILTERED</i>
When value of parameter option is <i>HALFDUPLEX_OPTION</i>	
value1	<i>ENET_CARRIERSENSE_ENABLE / ENET_CARRIERSENSE_DISABLE</i>
value2	<i>ENET_RECEIVEOWN_ENABLE / ENET_RECEIVEOWN_DISABLE</i>
value3	<i>ENET_RETRYTRANSMISSION_ENABLE / ENET_RETRYTRANSMISSION_DISABLE</i>
value4	<i>ENET_BACKOFFLIMIT_10 / ENET_BACKOFFLIMIT_8 / ENET_BACKOFFLIMIT_4 / ENET_BACKOFFLIMIT_1</i>
value5	<i>ENET_DEFERRALCHECK_ENABLE / ENET_DEFERRALCHECK_DISABLE</i>
When value of parameter option is <i>TIMER_OPTION</i>	
value1	<i>ENET_WATCHDOG_ENABLE / ENET_WATCHDOG_DISABLE</i>
value2	<i>ENET_JABBER_ENABLE / ENET_JABBER_DISABLE</i>
When value of parameter option is <i>INTERFRAMEGAP_OPTION</i>	
value1	<i>ENET_INTERFRAMEGAP_96BIT / ENET_INTERFRAMEGAP_88BIT / ENET_INTERFRAMEGAP_80BIT / ENET_INTERFRAMEGAP_72BIT / ENET_INTERFRAMEGAP_64BIT / ENET_INTERFRAMEGAP_56BIT / ENET_INTERFRAMEGAP_48BIT / ENET_INTERFRAMEGAP_40BIT</i>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* config DMA option of the ENET */
enet_initpara_reset();

enet_initpara_config(DMA_OPTION, ENET_FLUSH_RXFRAME_ENABLE | ENET_SECONDFRAME_OPT_ENABLE | ENET_NORMAL_DESCRIPTOR);

```

enet_init

The description of enet_init is shown as below:

Table 3-225. Function enet_init

Function name	enet_init
Function prototype	ErrStatus enet_init(enet_mediemode_enum mediemode, enetcsumconf_enum checksum, enet_frmrecept_enum recept);
Function descriptions	initialize ENET peripheral with generally concerned parameters and the less cared parameters
Precondition	enet_deinit ()
The called functions	enet_phy_config /enet_phy_write_read
Input parameter{in}	
mediemode	PHY mode and mac loopback configurations, only one parameter can be selected
<i>ENET_AUTO_NEGOTIATION</i>	PHY auto negotiation
<i>ENET_100M_FULLDUPLEX</i>	100Mbit/s, full-duplex
<i>ENET_100M_HALFDUPLEX</i>	100Mbit/s, half-duplex
<i>ENET_10M_FULLDUPLEX</i>	10Mbit/s, full-duplex
<i>ENET_10M_HALFDUPLEX</i>	10Mbit/s, half-duplex
<i>ENET_LOOPBACKMODE</i>	MAC in loopback mode at the MII
Input parameter{in}	
checksum	IP frame checksum offload function, only one parameter can be selected
<i>ENET_NO_AUTOCHECKSUM</i>	disable IP frame checksum function
<i>ENET_AUTOCHECKSUM_DROP_FAILFRAMES</i>	enable IP frame checksum function
<i>ENET_AUTOCHECKSUM_ACCEPT_FAILFRAMES</i>	enable IP frame checksum function, and the received frame with only payload error but no other errors will not be dropped

Input parameter{in}	
receipt	frame filter function, only one parameter can be selected
ENET_PROMISCUOUS_MODE	promiscuous mode enabled
ENET_RECEIVEALL	all received frame are forwarded to application
ENET_BROADCAST_FRAMES_PASS	the address filters pass all received broadcast frames
ENET_BROADCAST_FRAMES_DROP	the address filters filter all incoming broadcast frames
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```

/* initialize ENET peripheral */

ErrStatus enet_init_status;

enet_init_status = enet_init(ENET_AUTO_NEGOTIATION, ENET_AUTOCHECKSUM_DR
OP_FAILFRAMES, ENET_BROADCAST_FRAMES_PASS);

```

enet_software_reset

The description of **enet_software_reset** is shown as below:

Table 3-226. Function enet_software_reset

Function name	enet_software_reset
Function prototype	ErrStatus enet_software_reset(void);
Function descriptions	reset all core internal registers located in CLK_TX and CLK_RX
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* reset all core internal registers located in CLK_TX and CLK_RX */

ErrStatus reval_state = ERROR;

reval_state = enet_software_reset();
```

enet_rxframe_size_get

The description of enet_rxframe_size_get is shown as below:

Table 3-227. Function enet_rxframe_size_get

Function name	enet_rxframe_size_get
Function prototype	uint32_t enet_rxframe_size_get(void);
Function descriptions	check receive frame valid and return frame size
Precondition	-
The called functions	enet_rxframe_drop()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	size of received frame 0x0 - 0x3FFF

Example:

```
/* check receive frame valid */

uint32_t reval;

reval = enet_rxframe_size_get();
```

enet_descriptors_chain_init

The description of enet_descriptors_chain_init is shown as below:

Table 3-228. Function enet_descriptors_chain_init

Function name	enet_descriptors_chain_init
Function prototype	void enet_descriptors_chain_init(enet_dmadirection_enum direction);
Function descriptions	initialize the DMA Tx/Rx descriptors's parameters in chain mode
Precondition	-
The called functions	-
Input parameter{in}	
direction	the descriptors which users want to init only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the DMA Tx/Rx descriptors's parameters in chain mode */
enet_descriptors_chain_init(ENET_DMA_TX);
```

enet_descriptors_ring_init

The description of enet_descriptors_ring_init is shown as below:

Table 3-229. Function enet_descriptors_ring_init

Function name	enet_descriptors_ring_init
Function prototype	void enet_descriptors_ring_init(enet_dmadirection_enum direction);
Function descriptions	initialize the DMA Tx/Rx descriptors's parameters in ring mode
Precondition	-
The called functions	-
Input parameter{in}	
direction	the descriptors which users want to init only one parameter can be selected which is shown as below

<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the DMA Tx/Rx descriptors's parameters in ring mode */

enet_descriptors_ring_init(ENET_DMA_TX);
```

enet_frame_receive

The description of enet_frame_receive is shown as below:

Table 3-230. Function enet_frame_receive

Function name	enet_frame_receive
Function prototype	ErrStatus enet_frame_receive(uint8_t *buffer, uint32_t bufsize);
Function descriptions	handle current received frame data to application buffer
Precondition	-
The called functions	-
Input parameter{in}	
bufsize	the size of buffer which is the parameter in function, (0 -- 1524)
Output parameter{out}	
buffer	pointer to the received frame data if the input is NULL, user should copy data in application by himself
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* transfer received frame data to application buffer */

uint8_t data_buffer[1500];

uint32_t data_size;
```

```
enet_frame_receive(data_buffer, &data_size);
```

enet_frame_transmit

The description of enet_frame_transmit is shown as below:

Table 3-231. Function enet_frame_transmit

Function name	enet_frame_transmit
Function prototype	ErrStatus enet_frame_transmit(uint8_t *buffer, uint32_t length);
Function descriptions	handle application buffer data to transmit it
Precondition	-
The called functions	-
Input parameter{in}	
buffer	pointer to the frame data to be transmitted if the input is NULL, user should handle the data in application by himself
Input parameter{in}	
length	the length of frame data to be transmitted, (0 -- 1524)
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* transfer buffer data of application */

uint8_t data_buffer[1500];

uint32_t data_size = 800;

enet_frame_transmit (data_buffer, data_size);
```

enet_transmit_checksum_config

The description of enet_transmit_checksum_config is shown as below:

Table 3-232. Function enet_transmit_checksum_config

Function name	enet_transmit_checksum_config
Function prototype	void enet_transmit_checksum_config(enet_descriptors_struct *desc, uint32_t checksum);

Function descriptions	configure the transmit IP frame checksum offload calculation and insertion
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to configure, the structure members can refer to Structure enet descriptors struct
Input parameter{in}	
checksum	IP frame checksum configuration only one parameter can be selected which is shown as below
<i>ENET_CHECKSUM_DISABLE</i>	checksum insertion disabled
<i>ENET_CHECKSUM_IPV4HEADER</i>	only IP header checksum calculation and insertion are enabled
<i>ENET_CHECKSUM_TCUPDPICMP_SEGMENT</i>	TCP/UDP/ICMP checksum insertion calculated but pseudo-header
<i>ENET_CHECKSUM_TCUPDPICMP_FULL</i>	TCP/UDP/ICMP checksum insertion fully calculated
Output parameter{out}	
Return value	

Example:

```

/* configure the transmit IP frame checksum offload calculation and insertion */

enet_descriptors_struct rx_desc;

enet_transmit_checksum_config(rx_desc, ENET_CHECKSUM_TCUPDPICMP_FULL);

```

enet_enable

The description of `enet_enable` is shown as below:

Table 3-233. Function `enet_enable`

Function name	<code>enet_enable</code>
----------------------	--------------------------

Function prototype	void enet_enable(void);
Function descriptions	ENET Tx and Rx function enable (include MAC and DMA module)
Precondition	-
The called functions	enet_tx_enable() /enet_rx_enable()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the ENET */
enet_enable();
```

enet_disable

The description of enet_disable is shown as below:

Table 3-234. Function enet_disable

Function name	enet_disable
Function prototype	void enet_disable(void);
Function descriptions	ENET Tx and Rx function disable (include MAC and DMA module)
Precondition	-
The called functions	enet_tx_disable() /enet_rx_disable()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the ENET */
```

```
enet_disable();
```

enet_mac_address_set

The description of enet_mac_address_set is shown as below:

Table 3-235. Function enet_mac_address_set

Function name	enet_mac_address_set
Function prototype	void enet_mac_address_set(enet_macaddress_enum mac_addr, uint8_t paddr[]);
Function descriptions	configure MAC address
Precondition	-
The called functions	-
Input parameter{in}	
mac_addr	select which MAC address will be set only one parameter can be selected which is shown as below
<i>ENET_MAC_ADDRESS_S0</i>	set MAC address 0 filter
<i>ENET_MAC_ADDRESS_S1</i>	set MAC address 1 filter
<i>ENET_MAC_ADDRESS_S2</i>	set MAC address 2 filter
<i>ENET_MAC_ADDRESS_S3</i>	set MAC address 3 filter
Input parameter{in}	
paddr	the buffer pointer which stores the MAC address little-ending store, such as MAC address is aa:bb:cc:dd:ee:22, the buffer is {22, ee, dd, cc, bb, aa}
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* config mac address */

netif->hwaddr[0] = 0x02;

netif->hwaddr[1] = 0xaa;

netif->hwaddr[2] = 0xbb;

netif->hwaddr[3] = 0xcc;

netif->hwaddr[4] = 0xdd;

netif->hwaddr[5] = 0xee;

enet_mac_address_set(ENET_MAC_ADDRESS0, netif->hwaddr);

```

enet_mac_address_get

The description of enet_mac_address_get is shown as below:

Table 3-236. Function enet_mac_address_get

Function name	enet_mac_address_get
Function prototype	void enet_mac_address_get(enet_macaddress_enum mac_addr, uint8_t paddr[]);
Function descriptions	get MAC address
Precondition	-
The called functions	-
Input parameter{in}	
mac_addr	select which MAC address will be set only one parameter can be selected which is shown as below
ENET_MAC_ADDRESS0	set MAC address 0 filter
ENET_MAC_ADDRESS1	set MAC address 1 filter
ENET_MAC_ADDRESS2	set MAC address 2 filter
ENET_MAC_ADDRESS3	set MAC address 3 filter
Output parameter{out}	
paddr	the buffer pointer which stores the MAC address

	little-ending store, such as MAC address is aa:bb:cc:dd:ee:22, the buffer is {22, ee, dd, cc, bb, aa}
Return value	
-	-

Example:

```

/* get mac address */

enet_mac_address_get (ENET_MAC_ADDRESS0, netif->hwaddr);

```

enet_flag_get

The description of enet_flag_get is shown as below:

Table 3-237. Function enet_flag_get

Function name	enet_flag_get
Function prototype	FlagStatus enet_flag_get(enet_flag_enum enet_flag);
Function descriptions	get the ENET MAC/MSC/PTP/DMA status flag
Precondition	-
The called functions	-
Input parameter{in}	
enet_flag	ENET status flag, refer to enet_flag_enum only one parameter can be selected which is shown as below
<i>ENET_MAC_FLAG_MP KR</i>	magic packet received flag
<i>ENET_MAC_FLAG_W UFR</i>	wakeup frame received flag
<i>ENET_MAC_FLAG_FL OWCONTROL</i>	flow control status flag
<i>ENET_MAC_FLAG_W UM</i>	WUM status flag
<i>ENET_MAC_FLAG_MS C</i>	MSC status flag
<i>ENET_MAC_FLAG_MS CR</i>	MSC receive status flag
<i>ENET_MAC_FLAG_MS</i>	MSC transmit status flag

<i>CT</i>	
<i>ENET_MAC_FLAG_TM</i> <i>ST</i>	time stamp trigger status flag
<i>ENET_PTP_FLAG_TS</i> <i>SCO</i>	timestamp second counter overflow flag
<i>ENET_PTP_FLAG_TT</i> <i>M</i>	target time match flag
<i>ENET_MSC_FLAG_RF</i> <i>CE</i>	received frames CRC error flag
<i>ENET_MSC_FLAG_RF</i> <i>AE</i>	received frames alignment error flag
<i>ENET_MSC_FLAG_RG</i> <i>UF</i>	received good unicast frames flag
<i>ENET_MSC_FLAG_TG</i> <i>FSC</i>	transmitted good frames single collision flag
<i>ENET_MSC_FLAG_TG</i> <i>FMSC</i>	transmitted good frames more single collision flag
<i>ENET_MSC_FLAG_TG</i> <i>F</i>	transmitted good frames flag
<i>ENET_DMA_FLAG_TS</i>	transmit status flag
<i>ENET_DMA_FLAG_TP</i> <i>S</i>	transmit process stopped status flag
<i>ENET_DMA_FLAG_TB</i> <i>U</i>	transmit buffer unavailable status flag
<i>ENET_DMA_FLAG_TJ</i> <i>T</i>	transmit jabber timeout status flag
<i>ENET_DMA_FLAG_RO</i>	receive overflow status flag
<i>ENET_DMA_FLAG_TU</i>	transmit underflow status flag
<i>ENET_DMA_FLAG_RS</i>	receive status flag
<i>ENET_DMA_FLAG_RB</i> <i>U</i>	receive buffer unavailable status flag
<i>ENET_DMA_FLAG_RP</i> <i>S</i>	receive process stopped status flag

<i>ENET_DMA_FLAG_R</i>	receive watchdog timeout status flag
<i>ENET_DMA_FLAG_ET</i>	early transmit status flag
<i>ENET_DMA_FLAG_FB</i>	fatal bus error status flag
<i>ENET_DMA_FLAG_ER</i>	early receive status flag
<i>ENET_DMA_FLAG_AI</i>	abnormal interrupt summary flag
<i>ENET_DMA_FLAG_NI</i>	normal interrupt summary flag
<i>ENET_DMA_FLAG_EB</i> <i>_DMA_ERROR</i>	DMA error flag
<i>ENET_DMA_FLAG_EB</i> <i>_TRANSFER_ERROR</i>	transfer error flag
<i>ENET_DMA_FLAG_EB</i> <i>_ACCESS_ERROR</i>	access error flag
<i>ENET_DMA_FLAG_MS</i>	MSC status flag
<i>ENET_DMA_FLAG_W</i>	WUM status flag
<i>ENET_DMA_FLAG_TS</i>	timestamp trigger status flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check whether the specified flag bit is set */
enet_flag_get (ENET_DMA_FLAG_RS);
```

enet_flag_clear

The description of enet_flag_clear is shown as below:

Table 3-238. Function enet_flag_clear

Function name	enet_flag_clear

Function prototype	void enet_flag_clear(enet_flag_clear_enum enet_flag);
Function descriptions	clear the ENET DMA status flag
Precondition	-
The called functions	-
Input parameter{in}	
enet_flag	ENET DMA flag clear, refer to enet_flag_clear_enum only one parameter can be selected which is shown as below
<i>ENET_DMA_FLAG_TS _CLR</i>	transmit status flag clear
<i>ENET_DMA_FLAG_TP S_CLR</i>	transmit process stopped status flag clear
<i>ENET_DMA_FLAG_TB U_CLR</i>	transmit buffer unavailable status flag clear
<i>ENET_DMA_FLAG_TJ T_CLR</i>	transmit jabber timeout status flag clear
<i>ENET_DMA_FLAG_RO _CLR</i>	receive overflow status flag clear
<i>ENET_DMA_FLAG_TU _CLR</i>	transmit underflow status flag clear
<i>ENET_DMA_FLAG_RS _CLR</i>	receive status flag clear
<i>ENET_DMA_FLAG_RB U_CLR</i>	receive buffer unavailable status flag clear
<i>ENET_DMA_FLAG_RP S_CLR</i>	receive process stopped status flag clear
<i>ENET_DMA_FLAG_RW WT_CLR</i>	receive watchdog timeout status flag clear
<i>ENET_DMA_FLAG_ET _CLR</i>	early transmit status flag clear
<i>ENET_DMA_FLAG_FB E_CLR</i>	fatal bus error status flag clear
<i>ENET_DMA_FLAG_ER _CLR</i>	early receive status flag clear

<i>ENET_DMA_FLAG_AI_CLR</i>	abnormal interrupt summary flag clear
<i>ENET_DMA_FLAG_NI_CLR</i>	normal interrupt summary flag clear
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the specified flag bit */
enet_flag_clear(ENET_DMA_FLAG_RS_CLR);
```

enet_interrupt_enable

The description of enet_interrupt_enable is shown as below:

Table 3-239. Function enet_interrupt_enable

Function name	enet_interrupt_enable
Function prototype	void enet_interrupt_enable(enet_int_enum enet_int);
Function descriptions	enable ENET MAC/MSC/DMA interrupt
Precondition	-
The called functions	-
Input parameter{in}	
enet_int	ENET interrupt only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_WUM_IM</i>	WUM interrupt mask
<i>ENET_MAC_INT_TMS_TIM</i>	timestamp trigger interrupt mask
<i>ENET_MSC_INT_RFC_EIM</i>	received frame CRC error interrupt mask
<i>ENET_MSC_INT_RFA_EIM</i>	received frames alignment error interrupt mask

<i>ENET_MSC_INT_RGU FIM</i>	received good unicast frames interrupt mask
<i>ENET_MSC_INT_TGF SCIM</i>	transmitted good frames single collision interrupt mask
<i>ENET_MSC_INT_TGF MSCIM</i>	transmitted good frames more single collision interrupt mask
<i>ENET_MSC_INT_TGFI M</i>	transmitted good frames interrupt mask
<i>ENET_DMA_INT_TIE</i>	transmit interrupt enable
<i>ENET_DMA_INT_TPSI E</i>	transmit process stopped interrupt enable
<i>ENET_DMA_INT_TBUI E</i>	transmit buffer unavailable interrupt enable
<i>ENET_DMA_INT_TJTI E</i>	transmit jabber timeout interrupt enable
<i>ENET_DMA_INT_ROIE</i>	receive overflow interrupt enable
<i>ENET_DMA_INT_TUIE</i>	transmit underflow interrupt enable
<i>ENET_DMA_INT_RIE</i>	receive interrupt enable
<i>ENET_DMA_INT_RBUI E</i>	receive buffer unavailable interrupt enable
<i>ENET_DMA_INT_RPSI E</i>	receive process stopped interrupt enable
<i>ENET_DMA_INT_RWT IE</i>	receive watchdog timeout interrupt enable
<i>ENET_DMA_INT_ETIE</i>	early transmit interrupt enable
<i>ENET_DMA_INT_FBEI E</i>	fatal bus error interrupt enable
<i>ENET_DMA_INT_ERIE</i>	early receive interrupt enable
<i>ENET_DMA_INT_AIE</i>	abnormal interrupt summary enable
<i>ENET_DMA_INT_NIE</i>	normal interrupt summary enable
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* enable normal interrupt summary */
enet_interrupt_enable(ENET_DMA_INT_NIE);
```

enet_interrupt_disable

The description of enet_interrupt_disable is shown as below:

Table 3-240. Function enet_interrupt_disable

Function name	enet_interrupt_disable
Function prototype	void enet_interrupt_disable(enet_int_enum enet_int);
Function descriptions	disable ENET MAC/MSC/DMA interrupt
Precondition	-
The called functions	-
Input parameter{in}	
enet_int	ENET interrupt only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_WUM IM</i>	WUM interrupt mask
<i>ENET_MAC_INT_TMS TIM</i>	timestamp trigger interrupt mask
<i>ENET_MSC_INT_RFC EIM</i>	received frame CRC error interrupt mask
<i>ENET_MSC_INT_RFA EIM</i>	received frames alignment error interrupt mask
<i>ENET_MSC_INT_RGU FIM</i>	received good unicast frames interrupt mask
<i>ENET_MSC_INT_TGF SCIM</i>	transmitted good frames single collision interrupt mask
<i>ENET_MSC_INT_TGF MSCIM</i>	transmitted good frames more single collision interrupt mask

<i>ENET_MSC_INT_TGFI M</i>	transmitted good frames interrupt mask
<i>ENET_DMA_INT_TIE</i>	transmit interrupt enable
<i>ENET_DMA_INT_TPSI E</i>	transmit process stopped interrupt enable
<i>ENET_DMA_INT_TBUI E</i>	transmit buffer unavailable interrupt enable
<i>ENET_DMA_INT_TJTI E</i>	transmit jabber timeout interrupt enable
<i>ENET_DMA_INT_ROIE</i>	receive overflow interrupt enable
<i>ENET_DMA_INT_TUIE</i>	transmit underflow interrupt enable
<i>ENET_DMA_INT_RIE</i>	receive interrupt enable
<i>ENET_DMA_INT_RBUI E</i>	receive buffer unavailable interrupt enable
<i>ENET_DMA_INT_RPSI E</i>	receive process stopped interrupt enable
<i>ENET_DMA_INT_RWT IE</i>	receive watchdog timeout interrupt enable
<i>ENET_DMA_INT_ETIE</i>	early transmit interrupt enable
<i>ENET_DMA_INT_FBEI E</i>	fatal bus error interrupt enable
<i>ENET_DMA_INT_ERIE</i>	early receive interrupt enable
<i>ENET_DMA_INT_AIE</i>	abnormal interrupt summary enable
<i>ENET_DMA_INT_NIE</i>	normal interrupt summary enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable normal interrupt summary */
enet_interrupt_disable(ENET_DMA_INT_NIE);
```

enet_interrupt_flag_get

The description of enet_interrupt_flag_get is shown as below:

Table 3-241. Function enet_interrupt_flag_get

Function name	enet_interrupt_flag_get
Function prototype	FlagStatus enet_interrupt_flag_get(enet_int_flag_enum int_flag);
Function descriptions	get ENET MAC/MSC/DMA interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	ENET interrupt flag only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_FLA</i> <i>G_WUM</i>	WUM status flag
<i>ENET_MAC_INT_FLA</i> <i>G_MSC</i>	MSC status flag
<i>ENET_MAC_INT_FLA</i> <i>G_MSCR</i>	MSC receive status flag
<i>ENET_MAC_INT_FLA</i> <i>G_MSCT</i>	MSC transmit status flag
<i>ENET_MAC_INT_FLA</i> <i>G_TMST</i>	time stamp trigger status flag
<i>ENET_MSC_INT_FLA</i> <i>G_RFCE</i>	received frames CRC error flag
<i>ENET_MSC_INT_FLA</i> <i>G_RFAE</i>	received frames alignment error flag
<i>ENET_MSC_INT_FLA</i> <i>G_RGUF</i>	received good unicast frames flag
<i>ENET_MSC_INT_FLA</i> <i>G_TGFSC</i>	transmitted good frames single collision flag
<i>ENET_MSC_INT_FLA</i> <i>G_TGFMSC</i>	transmitted good frames more single collision flag
<i>ENET_MSC_INT_FLA</i>	transmitted good frames flag

<i>G_TGF</i>	
<i>ENET_DMA_INT_FLA_G_TS</i>	transmit status flag
<i>ENET_DMA_INT_FLA_G_TPS</i>	transmit process stopped status flag
<i>ENET_DMA_INT_FLA_G_TBU</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_FLA_G_TJT</i>	transmit jabber timeout status flag
<i>ENET_DMA_INT_FLA_G_RO</i>	receive overflow status flag
<i>ENET_DMA_INT_FLA_G_TU</i>	transmit underflow status flag
<i>ENET_DMA_INT_FLA_G_RS</i>	receive status flag
<i>ENET_DMA_INT_FLA_G_RBU</i>	receive buffer unavailable status flag
<i>ENET_DMA_INT_FLA_G_RPS</i>	receive process stopped status flag
<i>ENET_DMA_INT_FLA_G_RWT</i>	receive watchdog timeout status flag
<i>ENET_DMA_INT_FLA_G_ET</i>	early transmit status flag
<i>ENET_DMA_INT_FLA_G_FBE</i>	fatal bus error status flag
<i>ENET_DMA_INT_FLA_G_ER</i>	early receive status flag
<i>ENET_DMA_INT_FLA_G_AI</i>	abnormal interrupt summary flag
<i>ENET_DMA_INT_FLA_G_NI</i>	normal interrupt summary flag
<i>ENET_DMA_INT_FLA_G_MSC</i>	MSC status flag
<i>ENET_DMA_INT_FLA</i>	WUM status flag

<i>G_WUM</i>	
<i>ENET_DMA_INT_FLA</i> <i>G_TST</i>	timestamp trigger status flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check whether the specified flag bit is set or not */
enet_interrupt_flag_get(ENET_DMA_INT_FLAG_RS);
```

enet_interrupt_flag_clear

The description of `enet_interrupt_flag_clear` is shown as below:

Table 3-242. Function `enet_interrupt_flag_clear`

Function name	enet_interrupt_flag_clear
Function prototype	void enet_interrupt_flag_clear(enet_int_flag_clear_enum int_flag_clear);
Function descriptions	clear ENET DMA interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
<i>int_flag_clear</i>	clear ENET interrupt flag only one parameter can be selected which is shown as below
<i>ENET_DMA_INT_FLA</i> <i>G_TS_CLR</i>	transmit status flag
<i>ENET_DMA_INT_FLA</i> <i>G_TPS_CLR</i>	transmit process stopped status flag
<i>ENET_DMA_INT_FLA</i> <i>G_TBU_CLR</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_FLA</i> <i>G_TJT_CLR</i>	transmit jabber timeout status flag
<i>ENET_DMA_INT_FLA</i> <i>G_RO_CLR</i>	receive overflow status flag

<i>ENET_DMA_INT_FLAG_TU_CLR</i>	transmit underflow status flag
<i>ENET_DMA_INT_FLAG_RS_CLR</i>	receive status flag
<i>ENET_DMA_INT_FLAG_RBU_CLR</i>	receive buffer unavailable status flag
<i>ENET_DMA_INT_FLAG_RPS_CLR</i>	receive process stopped status flag
<i>ENET_DMA_INT_FLAG_RWT_CLR</i>	receive watchdog timeout status flag
<i>ENET_DMA_INT_FLAG_ET_CLR</i>	early transmit status flag
<i>ENET_DMA_INT_FLAG_FBE_CLR</i>	fatal bus error status flag
<i>ENET_DMA_INT_FLAG_ER_CLR</i>	early receive status flag
<i>ENET_DMA_INT_FLAG_AI_CLR</i>	abnormal interrupt summary flag
<i>ENET_DMA_INT_FLAG_NI_CLR</i>	normal interrupt summary flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear receive status flag bit */
enet_interrupt_flag_clear(ENET_DMA_INT_FLAG_RS);
```

enet_tx_enable

The description of enet_tx_enable is shown as below:

Table 3-243. Function enet_tx_enable

Function name	enet_tx_enable

Function prototype	void enet_tx_enable(void);
Function descriptions	ENET Tx function enable (include MAC and DMA module)
Precondition	-
The called functions	enet_txfifo_flush()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable transport function of MAC and DMA */
enet_tx_enable();
```

enet_tx_disable

The description of enet_tx_disable is shown as below:

Table 3-244. Function enet_tx_disable

Function name	enet_tx_disable
Function prototype	void enet_tx_disable(void);
Function descriptions	ENET Tx function disable (include MAC and DMA module)
Precondition	-
The called functions	enet_txfifo_flush()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable transport function of MAC and DMA */
```

```
enet_tx_disable();
```

enet_rx_enable

The description of enet_rx_enable is shown as below:

Table 3-245. Function enet_rx_enable

Function name	enet_rx_enable
Function prototype	void enet_rx_enable(void);
Function descriptions	ENET Rx function enable (include MAC and DMA module)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable reception function of MAC and DMA */
```

```
enet_rx_enable();
```

enet_rx_disable

The description of enet_rx_disable is shown as below:

Table 3-246. Function enet_rx_disable

Function name	enet_rx_disable
Function prototype	void enet_rx_disable(void);
Function descriptions	ENET Rx function disable (include MAC and DMA module)
Precondition	-
The called functions	-
Input parameter{in}	

-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable reception function of MAC and DMA */

enet_rx_disable();
```

enet_registers_get

The description of enet_registers_get is shown as below:

Table 3-247. Function enet_registers_get

Function name	enet_registers_get
Function prototype	void enet_registers_get(enet_registers_type_enum type, uint32_t *preg, uint32_t num);
Function descriptions	put registers value into the application buffer
Precondition	-
The called functions	-
Input parameter{in}	
type	register type which will be get only one parameter can be selected which is shown as below
<i>ALL_MAC_REG</i>	get the registers within the offset scope range ENET_MAC_CFG to ENET_MAC_FCTH
<i>ALL_MSC_REG</i>	get the registers within the offset scope range ENET_MSC_CTL to ENET_MSC_RGUFUNCT
<i>ALL_PTP_REG</i>	get the registers within the offset scope range ENET_PTP_TSCTL to ENET_PTP_PPSCTL
<i>ALL_DMA_REG</i>	get the registers within the offset scope range ENET_DMA_BCTL to ENET_DMA_CRBADDR
Input parameter{in}	
num	the number of registers that the user want to get, (0 -- 54)

Output parameter{out}	
preg	the application buffer pointer for storing the register value
Return value	
-	-

Example:

```

/* get all mac registers value */

uint32_t register_buffer[5];

enet_registers_get(ALL_MAC_REG, 5, register_buffer);

```

enet_debug_status_get

The description of enet_debug_status_get is shown as below:

Table 3-248. Function enet_debug_status_get

Function name	enet_debug_status_get
Function prototype	uint32_t enet_debug_status_get(uint32_t mac_debug);
Function descriptions	get the enet debug status from the debug register
Precondition	-
The called functions	--
Input parameter{in}	
mac_debug	enet debug status only one parameter can be selected which is shown as below
<i>ENET_MAC_RECEIVE_R_NOT_IDLE</i>	MAC receiver is not in idle state
<i>ENET_RX_ASYNCRRONOUS_FIFO_STATE</i>	Rx asynchronous FIFO status
<i>ENET_RXFIFO_WRITING</i>	RxFIFO is doing write operation
<i>ENET_RXFIFO_READ_STATUS</i>	RxFIFO read operation status
<i>ENET_RXFIFO_STATE</i>	RxFIFO state
<i>ENET_MAC_TRANSMITTER_NOT_IDLE</i>	MAC transmitter is not in idle state

<i>ENET_MAC_TRANSMITTER_STATUS</i>	status of MAC transmitter
<i>ENET_PAUSE_CONDITION_STATUS</i>	pause condition status
<i>ENET_TXFIFO_READ_STATUS</i>	TxFIFO read operation status
<i>ENET_TXFIFO_WRITING</i>	TxFIFO is doing write operation
<i>ENET_TXFIFO_NOT_EMPTY</i>	TxFIFO is not empty
<i>ENET_TXFIFO_FULL</i>	TxFIFO is full
Output parameter{out}	
-	-
Return value	
uint32_t	value of the status users want to get

Example:

```

/* get debug message of RxFIFO state */

uint32_t debug_value;

debug_value = enet_debug_status_get (ENET_RXFIFO_STATE);

```

enet_address_filter_enable

The description of **enet_address_filter_enable** is shown as below:

Table 3-249. Function enet_address_filter_enable

Function name	enet_address_filter_enable
Function prototype	void enet_address_filter_enable(enet_macaddress_enum mac_addr);
Function descriptions	enable the MAC address filter
Precondition	-
The called functions	--
Input parameter{in}	
mac_addr	select which MAC address will be enable only one parameter can be selected which is shown as below

<i>ENET_MAC_ADDRESS1</i>	enable MAC address 1 filter
<i>ENET_MAC_ADDRESS2</i>	enable MAC address 2 filter
<i>ENET_MAC_ADDRESS3</i>	enable MAC address 3 filter
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the MAC address 1 filter */
enet_address_filter_enable(ENET_MAC_ADDRESS1);
```

enet_address_filter_disable

The description of `enet_address_filter_disable` is shown as below:

Table 3-250. Function `enet_address_filter_disable`

Function name	enet_address_filter_disable
Function prototype	void enet_address_filter_disable(enet_macaddress_enum mac_addr);
Function descriptions	disable the MAC address filter
Precondition	-
The called functions	-
Input parameter{in}	
mac_addr	select which MAC address will be enable only one parameter can be selected which is shown as below
<i>ENET_MAC_ADDRESS1</i>	enable MAC address 1 filter
<i>ENET_MAC_ADDRESS2</i>	enable MAC address 2 filter
<i>ENET_MAC_ADDRESS3</i>	enable MAC address 3 filter
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable the MAC address 1 filter */

enet_address_filter_disable(ENET_MAC_ADDRESS1);
```

enet_address_filter_config

The description of `enet_address_filter_config` is shown as below:

Table 3-251. Function `enet_address_filter_config`

Function name	enet_address_filter_config
Function prototype	void enet_address_filter_config(enet_macaddress_enum mac_addr, uint32_t addr_mask, uint32_t filter_type);
Function descriptions	configure the MAC address filter
Precondition	-
The called functions	-
Input parameter{in}	
mac_addr	select which MAC address will be enable only one parameter can be selected which is shown as below
<i>ENET_MAC_ADDRESS1</i>	enable MAC address 1 filter
<i>ENET_MAC_ADDRESS2</i>	enable MAC address 2 filter
<i>ENET_MAC_ADDRESS3</i>	enable MAC address 3 filter
Input parameter{in}	
addr_mask	select which MAC address bytes will be mask one or more parameters can be selected which are shown as below
<i>ENET_ADDRESS_MASK_BYTE0</i>	mask ENET_MAC_ADDR1L[7:0] bits
<i>ENET_ADDRESS_MASK_BYTE1</i>	mask ENET_MAC_ADDR1L[15:8] bits

<i>ENET_ADDRESS_MASK_BYTE2</i>	mask ENET_MAC_ADDR1L[23:16] bits
<i>ENET_ADDRESS_MASK_BYTE3</i>	mask ENET_MAC_ADDR1L [31:24] bits
<i>ENET_ADDRESS_MASK_BYTE4</i>	mask ENET_MAC_ADDR1H [7:0] bits
<i>ENET_ADDRESS_MASK_BYTE5</i>	mask ENET_MAC_ADDR1H [15:8] bits
Input parameter{in}	
filter_type	select which MAC address filter type will be selected only one parameter can be selected which is shown as below
<i>ENET_ADDRESS_FILTER_ER_SA</i>	The MAC address is used to compared with the SA field of the received frame
<i>ENET_ADDRESS_FILTER_ER_DA</i>	The MAC address is used to compared with the DA field of the received frame
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config the MAC address 1 filter */

enet_address_filter_config(ENET_MAC_ADDRESS1, ENET_ADDRESS_MASK_BYTE0 |  
ENET_ADDRESS_MASK_BYTE1 | ENET_ADDRESS_MASK_BYTE2, ENET_ADDRESS_FILTER_DA);
```

enet_phy_config

The description of enet_phy_config is shown as below:

Table 3-252. Function enet_phy_config

Function name	enet_phy_config
Function prototype	ErrStatus enet_phy_config(void);
Function descriptions	PHY interface configuration (configure SMI clock and reset PHY chip)
Precondition	-

The called functions	rcu_clock_freq_get()/enet_phy_write_read()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* config PHY interface */

enet_phy_config();
```

enet_phy_write_read

The description of enet_phy_write_read is shown as below:

Table 3-253. Function enet_phy_write_read

Function name	enet_phy_write_read
Function prototype	ErrStatus enet_phy_write_read(enet_phydirection_enum direction, uint16_t phy_address, uint16_t phy_reg, uint16_t *pvalue);
Function descriptions	write to / read from a PHY register
Precondition	-
The called functions	-
Input parameter{in}	
direction	only one parameter can be selected which is shown as below
<i>ENET_PHY_WRITE</i>	write data to phy register
<i>ENET_PHY_READ</i>	read data from phy register
Input parameter{in}	
phy_address	0x0 - 0x1F
Input parameter{in}	
phy_reg	0x0 - 0x1F
Input parameter{in}	

pvalue	the value will be written to the PHY register in ENET_PHY_WRITE direction
Output parameter{out}	
pvalue	the value will be read from the PHY register in ENET_PHY_READ direction
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```

/* write 0 to PHY BCR register */

uint16_t temp_phy = 0U;

phy_state = enet_phy_write_read(ENET_PHY_WRITE, PHY_ADDRESS, PHY_REG_BCR,
&temp_phy);

```

enet_phyloopback_enable

The description of `enet_phyloopback_enable` is shown as below:

Table 3-254. Function `enet_phyloopback_enable`

Function name	enet_phyloopback_enable
Function prototype	ErrStatus enet_phyloopback_enable(void);
Function descriptions	enable the loopback function of PHY chip
Precondition	-
The called functions	enet_phy_write_read()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```

/* enable the loopback function of PHY chip */

ErrStatus phy_state = ERROR;

phy_state = enet_phyloopback_enable();

```

enet_phyloopback_disable

The description of enet_phyloopback_disable is shown as below:

Table 3-255. Function enet_phyloopback_disable

Function name	enet_phyloopback_disable
Function prototype	ErrStatus enet_phyloopback_disable(void);
Function descriptions	disable the loopback function of PHY chip
Precondition	-
The called functions	enet_phy_write_read
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* disable the loopback function of PHY chip */
```

```
ErrStatus phy_state = ERROR;
```

```
phy_state = enet_phyloopback_disable();
```

enet_forward_feature_enable

The description of enet_forward_feature_enable is shown as below:

Table 3-256. Function enet_forward_feature_enable

Function name	enet_forward_feature_enable
Function prototype	void enet_forward_feature_enable(uint32_t feature);
Function descriptions	enable ENET forward feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET forward mode

	one or more parameters can be selected which are shown as below
<i>ENET_AUTO_PADCR_C_DROP</i>	the function of the MAC strips the Pad/FCS field on received frames
<i>ENET_TYPEFRAME_C_RC_DROP</i>	the function that FCS field(last 4 bytes) of frame will be dropped before forwarding
<i>ENET_FORWARD_ER_RFRAMES</i>	the function that all frame received with error except runt error are forwarded to memory
<i>ENET_FORWARD_UNDERSZ_GOODFRAMES</i>	the function that forwarding undersized good frames
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the function that forwarding undersized good frames */
enet_forward_feature_enable(ENET_FORWARD_UNDERSZ_GOODFRAMES);
```

enet_forward_feature_disable

The description of `enet_forward_feature_disable` is shown as below:

Table 3-257. Function `enet_forward_feature_disable`

Function name	enet_forward_feature_disable
Function prototype	void enet_forward_feature_disable(uint32_t feature);
Function descriptions	disable ENET forward feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET forward mode one or more parameters can be selected which are shown as below
<i>ENET_AUTO_PADCR_C_DROP</i>	the function of the MAC strips the Pad/FCS field on received frames

<i>ENET_TYPEFRAME_C RC_DROP</i>	the function that FCS field(last 4 bytes) of frame will be dropped before forwarding
<i>ENET_FORWARD_ER RFRAMES</i>	the function that all frame received with error except runt error are forwarded to memory
<i>ENET_FORWARD_UN DERSZ_GOODFRAME S</i>	the function that forwarding undersized good frames
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the function that forwarding undersized good frames */
enet_forward_feature_enable(ENET_FORWARD_UNDERSZ_GOODFRAMES);
```

enet_fliter_feature_enable

The description of **enet_fliter_feature_enable** is shown as below:

Table 3-258. Function enet_fliter_feature_enable

Function name	enet_fliter_feature_enable
Function prototype	void enet_fliter_feature_enable(uint32_t feature);
Function descriptions	enable ENET fliter feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET fliter mode one or more parameters can be selected which are shown as below
<i>ENET_SRC_FILTER</i>	filter source address function
<i>ENET_SRC_FILTER_INVERSE</i>	inverse source address filtering result function
<i>ENET_DEST_FILTER_INVERSE</i>	inverse DA filtering result function

<i>ENET_MULTICAST_FILTER_PASS</i>	pass all multicast frames function
<i>ENET_MULTICAST_FILTER_HASH_MODE</i>	HASH multicast filter function
<i>ENET_UNICAST_FILTER_HASH_MODE</i>	HASH unicast filter function
<i>ENET_FILTER_MODE_EITHER</i>	HASH or perfect filter function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable filter source address function */
enet_filter_feature_enable(ENET_SRC_FILTER);
```

enet_filter_feature_disable

The description of `enet_filter_feature_disable` is shown as below:

Table 3-259. Function `enet_filter_feature_disable`

Function name	enet_filter_feature_disable
Function prototype	void enet_filter_feature_disable(uint32_t feature);
Function descriptions	disable ENET filter feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET filter mode one or more parameters can be selected which are shown as below
<i>ENET_SRC_FILTER</i>	filter source address function
<i>ENET_SRC_FILTER_INVERSE</i>	inverse source address filtering result function
<i>ENET_DEST_FILTER</i>	inverse DA filtering result function

<i>INVERSE</i>	
<i>ENET_MULTICAST_FILTER_MODE_PASS</i>	pass all multicast frames function
<i>ENET_MULTICAST_FILTER_MODE_HASH_MODE</i>	HASH multicast filter function
<i>ENET_UNICAST_FILTER_MODE_HASH_MODE</i>	HASH unicast filter function
<i>ENET_FILTER_MODE_EITHER</i>	HASH or perfect filter function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable filter source address function */
enet_filter_feature_enable(ENET_SRC_FILTER);
```

enet_pauseframe_generate

The description of **enet_pauseframe_generate** is shown as below:

Table 3-260. Function enet_pauseframe_generate

Function name	enet_pauseframe_generate
Function prototype	ErrStatus enet_pauseframe_generate(void);
Function descriptions	generate the pause frame, ENET will send pause frame after enable transmit flow control this function only use in full-duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

ErrStatus	ERROR or SUCCESS
------------------	------------------

Example:

```
/* generate the pause frame */

ErrStatus reval;

reval = enet_pauseframe_generate();
```

enet_pauseframe_detect_config

The description of `enet_pauseframe_detect_config` is shown as below:

Table 3-261. Function `enet_pauseframe_detect_config`

Function name	enet_pauseframe_detect_config
Function prototype	void enet_pauseframe_detect_config(uint32_t detect);
Function descriptions	configure the pause frame detect type
Precondition	-
The called functions	-
Input parameter{in}	
detect	pause frame detect type only one parameter can be selected which is shown as below
<i>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDTECT</i>	besides the unique multicast address, MAC can also use the MAC0 address to detecting pause frame
<i>ENET_UNIQUE_PAUSEDTECT</i>	only the unique multicast address for pause frame which is specified in IEEE802.3 can be detected
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config the pause frame detect type as ENET_UNIQUE_PAUSEDTECT */

enet_pauseframe_detect_config(ENET_UNIQUE_PAUSEDTECT);
```

enet_pauseframe_config

The description of enet_pauseframe_config is shown as below:

Table 3-262. Function enet_pauseframe_config

Function name	enet_pauseframe_config
Function prototype	void enet_pauseframe_config(uint32_t pausetime, uint32_t pause_threshold);
Function descriptions	configure the pause frame parameters
Precondition	-
The called functions	-
Input parameter{in}	
pausetime	pause time in transmit pause control frame, (0 – 0xFFFF)
Input parameter{in}	
pause_threshold	the threshold of the pause timer for retransmitting frames automatically, this value must make sure to be less than configured pause time, only one parameter can be selected which is shown as below
<i>ENET_PAUSETIME_MINUS4</i>	pause time minus 4 slot times
<i>ENET_PAUSETIME_MINUS28</i>	pause time minus 28 slot times
<i>ENET_PAUSETIME_MINUS144</i>	pause time minus 144 slot times
<i>ENET_PAUSETIME_MINUS256</i>	pause time minus 256 slot times
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config pause time minus 4 slot times */

enet_pauseframe_config(30, ENET_PAUSETIME_MINUS4);
```

enet_flowcontrol_threshold_config

The description of enet_flowcontrol_threshold_config is shown as below:

Table 3-263. Function enet_flowcontrol_threshold_config

Function name	enet_flowcontrol_threshold_config
Function prototype	void enet_flowcontrol_threshold_config(uint32_t deactive, uint32_t active);
Function descriptions	configure the threshold of the flow control(deactive and active threshold)
Precondition	-
The called functions	-
Input parameter{in}	
deactive	the threshold of the deactive flow control, this value should always be less than active flow control value, only one parameter can be selected which is shown as below
<i>ENET_DEACTIVE_TH RESHOLD_256BYTES</i>	threshold level is 256 bytes
<i>ENET_DEACTIVE_TH RESHOLD_512BYTES</i>	threshold level is 512 bytes
<i>ENET_DEACTIVE_TH RESHOLD_768BYTES</i>	threshold level is 768 bytes
<i>ENET_DEACTIVE_TH RESHOLD_1024BYTES</i>	threshold level is 1024 bytes
<i>ENET_DEACTIVE_TH RESHOLD_1280BYTES</i>	threshold level is 1280 bytes
<i>ENET_DEACTIVE_TH RESHOLD_1536BYTES</i>	threshold level is 1536 bytes
<i>ENET_DEACTIVE_TH RESHOLD_1792BYTES</i>	threshold level is 1792 bytes
Input parameter{in}	
active	the threshold of the active flow control, only one parameter can be selected which is shown as below

<i>ENET_ACTIVE_THRESHOLD_256BYTES</i>	threshold level is 256 bytes
<i>ENET_ACTIVE_THRESHOLD_512BYTES</i>	threshold level is 512 bytes
<i>ENET_ACTIVE_THRESHOLD_768BYTES</i>	threshold level is 768 bytes
<i>ENET_ACTIVE_THRESHOLD_1024BYTES</i>	threshold level is 1024 bytes
<i>ENET_ACTIVE_THRESHOLD_1280BYTES</i>	threshold level is 1280 bytes
<i>ENET_ACTIVE_THRESHOLD_1536BYTES</i>	threshold level is 1536 bytes
<i>ENET_ACTIVE_THRESHOLD_1792BYTES</i>	threshold level is 1792 bytes
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the threshold of the flow control */
enet_flowcontrol_threshold_config(ENET_DEACTIVE_THRESHOLD_256BYTES, ENET_ACTIVE_THRESHOLD_256BYTES);
```

enet_flowcontrol_feature_enable

The description of `enet_flowcontrol_feature_enable` is shown as below:

Table 3-264. Function `enet_flowcontrol_feature_enable`

Function name	enet_flowcontrol_feature_enable
Function prototype	void enet_flowcontrol_feature_enable(uint32_t feature);
Function descriptions	enable ENET flow control feature
Precondition	-
The called functions	-
Input parameter{in}	

feature	the feature of ENET flow control mode one or more parameters can be selected which are shown as below
<i>ENET_ZERO_QUANT_A_PAUSE</i>	the automatic zero-quanta generation function
<i>ENET_TX_FLOWCONTROL</i>	the flow control operation in the MAC
<i>ENET_RX_FLOWCONTROL</i>	decoding function for the received pause frame and process it
<i>ENET_BACK_PRESSURE</i>	back pressure operation in the MAC(only use in half-duplex mode)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the flow control operation in the MAC */
enet_flowcontrol_feature_enable(ENET_ZERO_QUANTA_PAUSE);
```

enet_flowcontrol_feature_disable

The description of `enet_flowcontrol_feature_disable` is shown as below:

Table 3-265. Function `enet_flowcontrol_feature_disable`

Function name	enet_flowcontrol_feature_disable
Function prototype	void enet_flowcontrol_feature_disable(uint32_t feature);
Function descriptions	disable ENET flow control feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET flow control mode one or more parameters can be selected which are shown as below
<i>ENET_ZERO_QUANTA_PAUSE</i>	the automatic zero-quanta generation function

<i>ENET_TX_FLOWCONTROL</i>	the flow control operation in the MAC
<i>ENET_RX_FLOWCONTROL</i>	decoding function for the received pause frame and process it
<i>ENET_BACK_PRESSURE</i>	back pressure operation in the MAC(only use in half-duplex mode)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* disable the automatic zero-quanta generation function */
enet_flowcontrol_feature_disable(ENET_ZERO_QUANTA_PAUSE);

```

enet_dmaprocess_state_get

The description of `enet_dmaprocess_state_get` is shown as below:

Table 3-266. Function `enet_dmaprocess_state_get`

Function name	enet_dmaprocess_state_get
Function prototype	uint32_t enet_dmaprocess_state_get(enet_dmadirection_enum direction);
Function descriptions	get the dma transmit/receive process state
Precondition	-
The called functions	-
Input parameter{in}	
direction	choose the direction of DMA process which users want to resume only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA transmit process
<i>ENET_DMA_RX</i>	DMA receive process
Output parameter{out}	
-	-
Return value	

uint32_t	state of dma process, the value range shows below: ENET_RX_STATE_STOPPED / ENET_RX_STATE_FETCHING / ENET_RX_STATE_WAITING / ENET_RX_STATE_SUSPENDED / ENET_RX_STATE_CLOSING / ENET_RX_STATE_QUEUEING / ENET_TX_STATE_STOPPED / ENET_TX_STATE_FETCHING / ENET_TX_STATE_WAITING / ENET_TX_STATE_READING / ENET_TX_STATE_SUSPENDED / ENET_TX_STATE_CLOSING
-----------------	---

Example:

```

/* get the dma receive process state */

uint32_t eval;

eval = enet_dmaprocess_state_get(ENET_DMA_RX);

if(ENET_RX_STATE_SUSPENDED == eval){

    do...

}

```

enet_dmaprocess_resume

The description of `enet_dmaprocess_resume` is shown as below:

Table 3-267. Function `enet_dmaprocess_resume`

Function name	enet_dmaprocess_resume
Function prototype	void enet_dmaprocess_resume(enet_dmadirection_enum direction);
Function descriptions	poll the DMA transmission/reception enable by writing any value to the ENET_DMA_TPEN/ENET_DMA_RPEN register, this will make the DMA to resume transmission/reception
Precondition	-
The called functions	-
Input parameter{in}	
direction	choose the direction of DMA process which users want to resume only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA transmit process
<i>ENET_DMA_RX</i>	DMA receive process
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* enable DMA receive process */
enet_dmaprocess_resume(ENET_DMA_RX);
```

enet_rxprocess_check_recovery

The description of enet_rxprocess_check_recovery is shown as below:

Table 3-268. Function enet_rxprocess_check_recovery

Function name	enet_rxprocess_check_recovery
Function prototype	void enet_rxprocess_check_recovery(void);
Function descriptions	check and recover the Rx process
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* check and recover the Rx process */
enet_rxprocess_check_recovery();
```

enet_txfifo_flush

The description of enet_txfifo_flush is shown as below:

Table 3-269. Function enet_txfifo_flush

Function name	enet_txfifo_flush
Function prototype	ErrStatus enet_txfifo_flush(void);
Function descriptions	flush the ENET transmit FIFO, and wait until the flush operation completes

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* flush the ENET transmit FIFO */
ErrStatus reval = ERROR;
reval = enet_txfifo_flush();
```

enet_current_desc_address_get

The description of `enet_current_desc_address_get` is shown as below:

Table 3-270. Function `enet_current_desc_address_get`

Function name	enet_current_desc_address_get
Function prototype	uint32_t enet_current_desc_address_get(enet_desc_reg_enum addr_get);
Function descriptions	get the transmit/receive address of current descriptor, or current buffer, or descriptor table
Precondition	-
The called functions	-
Input parameter{in}	
addr_get	choose the address which users want to get only one parameter can be selected which is shown as below
<i>ENET_RX_DESC_TAB LE</i>	the start address of the receive descriptor table
<i>ENET_RX_CURRENT_ DESC</i>	the start descriptor address of the current receive descriptor read by the RxDMA controller
<i>ENET_RX_CURRENT_ BUFFER</i>	the current receive buffer address being read by the RxDMA controller

<i>ENET_RX_DESC_TABLE</i>	the start address of the receive descriptor table
<i>ENET_RX_CURRENT_DESC</i>	the start descriptor address of the current receive descriptor read by the RxDMA controller
<i>ENET_RX_CURRENT_BUFFER</i>	the current receive buffer address being read by the RxDMA controller
Output parameter{out}	
-	-
Return value	
uint32_t	0- 0xFFFFFFFF

Example:

```
/* get the start address of the receive descriptor table */
uint32_t reval;
reval = enet_current_desc_address_get(ENET_RX_DESC_TABLE);
```

enet_desc_information_get

The description of `enet_desc_information_get` is shown as below:

Table 3-271. Function `enet_desc_information_get`

Function name	<code>enet_desc_information_get</code>
Function prototype	<code>uint32_t enet_desc_information_get(enet_descriptors_struct *desc, enet_descstate_enum info_get);</code>
Function descriptions	get the Tx or Rx descriptor information
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to get information, the structure members can refer to Structure <code>enet_descriptors_struct</code>
Input parameter{in}	
info_get	the descriptor information type which is selected only one parameter can be selected which is shown as below
<i>RXDESC_BUFFER_1_</i>	receive buffer 1 size

SIZE	
RXDESC_BUFFER_2_SIZE	receive buffer 2 size
RXDESC_FRAME_LENGTH	the byte length of the received frame that was transferred to the buffer
TXDESC_COLLISION_COUNT	the number of collisions occurred before the frame was transmitted
RXDESC_BUFFER_1_ADDR	the buffer1 address of the Rx frame
TXDESC_BUFFER_1_ADDR	the buffer1 address of the Tx frame
Output parameter{out}	
-	-
Return value	
uint32_t	descriptor information if value is 0xFFFFFFFFU, means the false input parameter

Example:

```
/* get the reception buffer 1 size */
uint32_t reval;
reval = enet_desc_information_get(rx_desc, RXDESC_BUFFER_1_SIZE);
```

enet_missed_frame_counter_get

The description of **enet_missed_frame_counter_get** is shown as below:

Table 3-272. Function enet_missed_frame_counter_get

Function name	enet_missed_frame_counter_get
Function prototype	void enet_missed_frame_counter_get(uint32_t *rxfifo_drop, uint32_t *rxdma_drop);
Function descriptions	get the number of missed frames during receiving
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
rxfifo_drop	pointer to the number of frames dropped by RxFIFO
Output parameter{out}	
rxdma_drop	pointer to the number of frames missed by the RxDMA controller
Return value	
-	-

Example:

```

/* get the number of missed frames during receiving */

uint32_t rxcnt, txcnt;

enet_missed_frame_counter_get(&rxcnt, &txcnt);

```

enet_desc_flag_get

The description of **enet_desc_flag_get** is shown as below:

Table 3-273. Function enet_desc_flag_get

Function name	enet_desc_flag_get
Function prototype	FlagStatus enet_desc_flag_get(enet_descriptors_struct *desc, uint32_t desc_flag);
Function descriptions	get the bit flag of ENET DMA descriptor
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to get flag, the structure members can refer to Structure enet_descriptors_struct
Input parameter{in}	
desc_flag (the value according to the parameter desc)	the bit flag of ENET DMA descriptor only one parameter can be selected which is shown as below
When type of parameter desc is TX	
ENET_TDES0_DB	deferred
ENET_TDES0_UFE	underflow error

<i>ENET_TDES0_EXD</i>	excessive deferral
<i>ENET_TDES0_VFRM</i>	VLAN frame
<i>ENET_TDES0_ECO</i>	excessive collision
<i>ENET_TDES0_LCO</i>	late collision
<i>ENET_TDES0_NCA</i>	no carrier
<i>ENET_TDES0_LCA</i>	loss of carrier
<i>ENET_TDES0_IPPE</i>	IP payload error
<i>ENET_TDES0_FRMF</i>	frame flushed
<i>ENET_TDES0_JT</i>	jabber timeout
<i>ENET_TDES0_ES</i>	error summary
<i>ENET_TDES0_IPHE</i>	IP header error
<i>ENET_TDES0_TTMSS</i>	transmit timestamp status
<i>ENET_TDES0_TCHM</i>	the second address chained mode
<i>ENET_TDES0_TERM</i>	transmit end of ring mode
<i>ENET_TDES0_TTSEN</i>	transmit timestamp function enable
<i>ENET_TDES0_DPAD</i>	disable adding pad
<i>ENET_TDES0_DCRC</i>	disable CRC
<i>ENET_TDES0_FSG</i>	first segment
<i>ENET_TDES0_LSG</i>	last segment
<i>ENET_TDES0_INTC</i>	interrupt on completion
<i>ENET_TDES0_DAV</i>	DAV bit
When type of parameter desc is RX	
<i>ENET_RDES0_PCERR</i>	payload checksum error
<i>ENET_RDES0_CERR</i>	CRC error
<i>ENET_RDES0_DBERR</i>	dribble bit error
<i>ENET_RDES0_RERR</i>	receive error
<i>ENET_RDES0_RWDT</i>	receive watchdog timeout
<i>ENET_RDES0_FRMT</i>	frame type

<i>ENET_RDES0_LCO</i>	late collision
<i>ENET_RDES0_IPHER_R</i>	IP frame header error
<i>ENET_RDES0_LDES</i>	last descriptor
<i>ENET_RDES0_FDES</i>	first descriptor
<i>ENET_RDES0_VTAG</i>	VLAN tag
<i>ENET_RDES0_OERR</i>	overflow error
<i>ENET_RDES0_LERR</i>	length error
<i>ENET_RDES0_SAFF</i>	SA filter fail
<i>ENET_RDES0_DERR</i>	descriptor error
<i>ENET_RDES0_ERRS</i>	error summary
<i>ENET_RDES0_DAFF</i>	destination address filter fail
<i>ENET_RDES0_DAV</i>	descriptor available
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the bit flag of ENET DMA descriptor */

FlagStatus reval;

reval = enet_desc_flag_get(p_txdesc, ENET_TDES0_TCHM);
```

enet_desc_flag_set

The description of `enet_desc_flag_set` is shown as below:

Table 3-274. Function `enet_desc_flag_set`

Function name	<code>enet_desc_flag_set</code>
Function prototype	<code>void enet_desc_flag_set(enet_descriptors_struct *desc, uint32_t desc_flag);</code>
Function descriptions	set the bit flag of ENET DMA descriptor
Precondition	-

The called functions		-
Input parameter{in}		
desc	the descriptor pointer which users want to set flag, the structure members can refer to <u>Structure enet_descriptors_struct</u>	
Input parameter{in}		
desc_flag (the value according to the parameter desc)	the bit flag of ENET DMA descriptor only one parameter can be selected which is shown as below	
When type of parameter desc is TX		
<i>ENET_TDES0_VFRM</i>	VLAN frame	
<i>ENET_TDES0_FRMF</i>	frame flushed	
<i>ENET_TDES0_TCHM</i>	the second address chained mode	
<i>ENET_TDES0_TERM</i>	transmit end of ring mode	
<i>ENET_TDES0_TTSEN</i>	transmit timestamp function enable	
<i>ENET_TDES0_DPAD</i>	disable adding pad	
<i>ENET_TDES0_DCRC</i>	disable CRC	
<i>ENET_TDES0_FSG</i>	first segment	
<i>ENET_TDES0_LSG</i>	last segment	
<i>ENET_TDES0_INTC</i>	interrupt on completion	
<i>ENET_TDES0_DAV</i>	DAV bit	
When type of parameter desc is RX		
<i>ENET_RDES0_DAV</i>	descriptor available	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

```
/* set VLAN frame bit flag of ENET DMA descriptor */
```

```
enet_desc_flag_set(p_txdesc, ENET_TDES0_VFRM);
```

enet_desc_flag_clear

The description of enet_desc_flag_clear is shown as below:

Table 3-275. Function enet_desc_flag_clear

Function name	enet_desc_flag_clear
Function prototype	void enet_desc_flag_clear(enet_descriptors_struct *desc, uint32_t desc_flag);
Function descriptions	clear the bit flag of ENET DMA descriptor
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to set flag, the structure members can refer to Structure enet descriptors struct
Input parameter{in}	
desc_flag (the value according to the parameter desc)	the bit flag of ENET DMA descriptor only one parameter can be selected which is shown as below
When type of parameter desc is TX	
<i>ENET_TDES0_VFRM</i>	VLAN frame
<i>ENET_TDES0_FRMF</i>	frame flushed
<i>ENET_TDES0_TCHM</i>	the second address chained mode
<i>ENET_TDES0_TERM</i>	transmit end of ring mode
<i>ENET_TDES0_TTSEN</i>	transmit timestamp function enable
<i>ENET_TDES0_DPAD</i>	disable adding pad
<i>ENET_TDES0_DCRC</i>	disable CRC
<i>ENET_TDES0_FSG</i>	first segment
<i>ENET_TDES0_LSG</i>	last segment
<i>ENET_TDES0_INTC</i>	interrupt on completion
<i>ENET_TDES0_DAV</i>	DAV bit
When type of parameter desc is RX	

<i>ENET_RDES0_DAV</i>	descriptor available
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear VLAN frame bit flag of ENET DMA descriptor */
enet_desc_flag_clear(p_txdesc, ENET_TDES0_VFRM);
```

enet_rx_desc_immediate_receive_complete_interrupt

The description of `enet_rx_desc_immediate_receive_complete_interrupt` is shown as below:

Table 3-276. Function `enet_rx_desc_immediate_receive_complete_interrupt`

Function name	enet_rx_desc_immediate_receive_complete_interrupt
Function prototype	void enet_rx_desc_immediate_receive_complete_interrupt(enet_descriptors_struct *desc);
Function descriptions	when receiving completed, set RS bit in ENET_DMA_STAT register will immediately set
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to get flag, the structure members can refer to Structure enet_descriptors_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set RS bit in ENET_DMA_STAT register immediately when receiving completed */
enet_rx_desc_immediate_receive_complete_interrupt(p_rxdesc);
```

enet_rx_desc_delay_receive_complete_interrupt

The description of enet_rx_desc_delay_receive_complete_interrupt is shown as below:

Table 3-277. Function enet_rx_desc_delay_receive_complete_interrupt

Function name	enet_rx_desc_delay_receive_complete_interrupt
Function prototype	void enet_rx_desc_delay_receive_complete_interrupt(enet_descriptors_struct *desc, uint32_t delay_time);
Function descriptions	when receiving completed, set RS bit in ENET_DMA_STAT register will be set after a configurable delay time
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to get flag, the structure members can refer to <u>Structure enet_descriptors_struct</u>
Input parameter{in}	
delay_time	delay a time of 256*delay_time HCLK(0x00000000 - 0x000000FF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* when receiving completed, RS bit in ENET_DMA_STAT register will be set after 256*16
HCLK */

enet_rx_desc_delay_receive_complete_interrupt(p_rxdesc, 0x00000010);
```

enet_rxframe_drop

The description of enet_rxframe_drop is shown as below:

Table 3-278. Function enet_rxframe_drop

Function name	enet_rxframe_drop
Function prototype	void enet_rxframe_drop(void);
Function descriptions	drop current receive frame

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* drop current receive frame */

enet_rxframe_drop( );
```

enet_dma_feature_enable

The description of enet_dma_feature_enable is shown as below:

Table 3-279. Function enet_dma_feature_enable

Function name	enet_dma_feature_enable
Function prototype	void enet_dma_feature_enable(uint32_t feature);
Function descriptions	enable DMA feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of DMA mode one or more parameters can be selected which are shown as below
<i>ENET_NO_FLUSH_RX FRAME</i>	RxDMA does not flushes frames function
<i>ENET_SECONDFRAM E_OPT</i>	TxDMA controller operate on second frame function
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable RxDMA does not flushes frames function */
enet_dma_feature_enable(ENET_NO_FLUSH_RXFRAME);
```

enet_dma_feature_disable

The description of enet_dma_feature_disable is shown as below:

Table 3-280. Function enet_dma_feature_disable

Function name	enet_dma_feature_disable
Function prototype	void enet_dma_feature_disable(uint32_t feature);
Function descriptions	disable DMA feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of DMA mode one or more parameters can be selected which are shown as below
<i>ENET_NO_FLUSH_RXFRAME</i>	RxDMA does not flushes frames function
<i>ENET_SECONDFRAME_OPT</i>	TxDMA controller operate on second frame function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RxDMA does not flushes frames function */
enet_dma_feature_disable(ENET_NO_FLUSH_RXFRAME);
```

enet_rx_desc_enhanced_status_get

The description of enet_rx_desc_enhanced_status_get is shown as below:

Table 3-281. Function enet_rx_desc_enhanced_status_get

Function name	enet_rx_desc_enhanced_status_get
Function prototype	uint32_t enet_rx_desc_enhanced_status_get(enet_descriptors_struct *desc, uint32_t desc_status);
Function descriptions	get the bit of extended status flag in ENET DMA descriptor
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to get the extended status flag, the structure members can refer to Structure enet descriptors struct
Input parameter{in}	
desc_status	desc_status: the extended status want to get only one parameter can be selected which is shown as below
<i>ENET_RDES4_IPPLDT</i>	IP frame payload type
<i>ENET_RDES4_IPHER_R</i>	IP frame header error
<i>ENET_RDES4_IPPLDE_RR</i>	IP frame payload error
<i>ENET_RDES4_IPCKS_B</i>	IP frame checksum bypassed
<i>ENET_RDES4_IPF4</i>	IP frame in version 4
<i>ENET_RDES4_IPF6</i>	IP frame in version 6
<i>ENET_RDES4_PTPMT</i>	PTP message type
<i>ENET_RDES4_PTPOE_F</i>	PTP on ethernet frame
<i>ENET_RDES4_PTPVF</i>	PTP version format
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* get the IP frame payload type in ENET DMA descriptor */

uint32_t status;

status = enet_rx_desc_enhanced_status_get(p_rxdesc, ENET_RDES4_IPPLDT);

```

enet_desc_select_enhanced_mode

The description of `enet_desc_select_enhanced_mode` is shown as below:

Table 3-282. Function `enet_desc_select_enhanced_mode`

Function name	enet_desc_select_enhanced_mode
Function prototype	void enet_desc_select_enhanced_mode(void);
Function descriptions	configure descriptor to work in enhanced mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure descriptor to work in enhanced mode */

enet_desc_select_enhanced_mode();

```

enet_ptp_enhanced_descriptors_chain_init

The description of `enet_ptp_enhanced_descriptors_chain_init` is shown as below:

Table 3-283. Function `enet_ptp_enhanced_descriptors_chain_init`

Function name	enet_ptp_enhanced_descriptors_chain_init
Function prototype	void enet_ptp_enhanced_descriptors_chain_init(enet_dmadirection_enum direction);
Function descriptions	initialize the DMA Tx/Rx descriptors's parameters in enhanced chain mode with ptp function
Precondition	-

The called functions	-
Input parameter{in}	
direction	the descriptors which users want to init, only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the DMA Tx descriptors's parameters in enhanced chain mode with ptp function */
enet_ptp_enhanced_descriptors_chain_init(ENET_DMA_TX);
```

enet_ptp_enhanced_descriptors_ring_init

The description of `enet_ptp_enhanced_descriptors_ring_init` is shown as below:

Table 3-284. Function `enet_ptp_enhanced_descriptors_ring_init`

Function name	enet_ptp_enhanced_descriptors_ring_init
Function prototype	void enet_ptp_enhanced_descriptors_ring_init(enet_dmadirection_enum direction);
Function descriptions	initialize the DMA Tx/Rx descriptors's parameters in enhanced ring mode with ptp function
Precondition	-
The called functions	-
Input parameter{in}	
direction	the descriptors which users want to init, only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* initialize the DMA Rx descriptors's parameters in enhanced ring mode with ptp function */
enet_ptp_enhanced_descriptors_ring_init(ENET_DMA_RX);
```

enet_ptpframe_receive_enhanced_mode

The description of enet_ptpframe_receive_enhanced_mode is shown as below:

Table 3-285. Function enet_ptpframe_receive_enhanced_mode

Function name	enet_ptpframe_receive_enhanced_mode
Function prototype	ErrStatus enet_ptpframe_receive_enhanced_mode(uint8_t *buffer, uint32_t_t bufsize, uint32_t timestamp[]);
Function descriptions	receive a packet data with timestamp values to application buffer, when the DMA is in enhanced mode
Precondition	-
The called functions	-
Input parameter{in}	
bufsize	the size of buffer which is the parameter in function
Output parameter{out}	
buffer	pointer to the application buffer
Output parameter{out}	
timestamp	pointer to the table which stores the timestamp high and low
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* receive a packet data with timestamp values to application buffer in DMA enhanced mode
*/
uint32_t rx_buffer[500];
uint32_t time_stamp[2];
```

```
ErrStatus status;
```

```
status = enet_ptpframe_receive_enhanced_mode(rx_buffer, 500, time_stamp);
```

enet_ptpframe_transmit_enhanced_mode

The description of enet_ptpframe_transmit_enhanced_mode is shown as below:

Table 3-286. Function enet_ptpframe_transmit_enhanced_mode

Function name	enet_ptpframe_transmit_enhanced_mode
Function prototype	ErrStatus enet_ptpframe_transmit_enhanced_mode(uint8_t *buffer, uint32_t length, uint32_t timestamp[]);
Function descriptions	send data with timestamp values in application buffer as a transmit packet, when the DMA is in enhanced mode
Precondition	-
The called functions	-
Input parameter{in}	
buffer	pointer on the application buffer note -- if the input is NULL, user should copy data in application by himself
Input parameter{in}	
length	the length of frame data to be transmitted
Output parameter{out}	
timestamp	pointer to the table which stores the timestamp high and low note -- if the input is NULL, timestamp is ignored
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* send data and timestamp values in application buffer as a transmit packet with DMA
enhanced mode */

uint32_t tx_buffer[500];

uint32_t time_stamp[2];

ErrStatus status;

status = enet_ptpframe_transmit_enhanced_mode(tx_buffer, 500, time_stamp);
```

enet_desc_select_normal_mode

The description of enet_desc_select_normal_mode is shown as below:

Table 3-287. Function enet_desc_select_normal_mode

Function name	enet_desc_select_normal_mode
Function prototype	void enet_desc_select_normal_mode(void);
Function descriptions	configure descriptor to work in normal mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure descriptor to work in normal mode */
enet_desc_select_normal_mode( );
```

enet_ptp_normal_descriptors_chain_init

The description of enet_ptp_normal_descriptors_chain_init is shown as below:

Table 3-288. Function enet_ptp_normal_descriptors_chain_init

Function name	enet_ptp_normal_descriptors_chain_init
Function prototype	void enet_ptp_normal_descriptors_chain_init(enet_dmadirection_enum direction, enet_descriptors_struct *desc_ptptab);
Function descriptions	initialize the DMA Tx/Rx descriptors's parameters in normal chain mode with PTP function
Precondition	-
The called functions	-
Input parameter{in}	
direction	the descriptors which users want to init

	only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors
Input parameter{in}	
desc_ptptab	pointer to the first descriptor address of PTP Rx descriptor table, the structure members can refer to <u>Structure enet_descriptors_struct</u>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the DMA Rx descriptors's parameters in normal chain mode with PTP function */
enet_descriptors_struct ptp_rxdesc_tab[ENET_RXBUF_NUM];
enet_ptp_normal_descriptors_chain_init(ENET_DMA_RX, ptp_rxdesc_tab);
```

enet_ptp_normal_descriptors_ring_init

The description of `enet_ptp_normal_descriptors_ring_init` is shown as below:

Table 3-289. Function `enet_ptp_normal_descriptors_ring_init`

Function name	enet_ptp_normal_descriptors_ring_init
Function prototype	void enet_ptp_normal_descriptors_ring_init(enet_dmadirction_enum direction, enet_descriptors_struct *desc_ptptab);
Function descriptions	initialize the DMA Tx/Rx descriptors's parameters in normal ring mode with PTP function
Precondition	-
The called functions	-
Input parameter{in}	
direction	the descriptors which users want to init only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors

Input parameter{in}	
desc_ptptab	pointer to the first descriptor address of PTP Rx descriptor table, the structure members can refer to <u>Structure enet_descriptors_struct</u>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize the DMA Rx descriptors's parameters in normal ring mode with PTP function */

enet_descriptors_struct ptp_rxdesc_tab[ENET_RXBUF_NUM];

enet_ptp_normal_descriptors_ring_init(ENET_DMA_RX, ptp_rxdesc_tab);

```

enet_ptpframe_receive_normal_mode

The description of enet_ptpframe_receive_normal_mode is shown as below:

Table 3-290. Function enet_ptpframe_receive_normal_mode

Function name	enet_ptpframe_receive_normal_mode
Function prototype	ErrStatus enet_ptpframe_receive_normal_mode(uint8_t *buffer, uint32_t bufsize, uint32_t timestamp[]);
Function descriptions	receive a packet data with timestamp values to application buffer, when the DMA is in normal mode
Precondition	-
The called functions	-
Input parameter{in}	
bufsize	the size of buffer which is the parameter in function
Output parameter{out}	
timestamp	pointer to the table which stores the timestamp high and low
Output parameter{out}	
buffer	pointer to the application buffer if the input is NULL, user should copy data in application by himself
Return value	

ErrStatus	ERROR or SUCCESS
------------------	------------------

Example:

```

/* receive a packet data with timestamp values to application buffer in DMA normal mode */

uint32_t rx_buffer[500];

uint32_t time_stamp[2];

ErrStatus status;

status = enet_ptpframe_receive_normal_mode(rx_buffer, 500, time_stamp);

```

enet_ptpframe_transmit_normal_mode

The description of `enet_ptpframe_transmit_normal_mode` is shown as below:

Table 3-291. Function `enet_ptpframe_transmit_normal_mode`

Function name	enet_ptpframe_transmit_normal_mode
Function prototype	ErrStatus enet_ptpframe_transmit_normal_mode(uint8_t *buffer, uint32_t length, uint32_t timestamp[]);
Function descriptions	send data with timestamp values in application buffer as a transmit packet, when the DMA is in normal mode
Precondition	-
The called functions	--
Input parameter{in}	
buffer	pointer on the application buffer if the input is NULL, user should copy data in application by himself
Input parameter{in}	
length	the length of frame data to be transmitted
Output parameter{out}	
timestamp	pointer to the table which stores the timestamp high and low if the input is NULL, timestamp is ignored
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```

/* send data and timestamp values in application buffer as a transmit packet with DMA normal mode */

```

```

uint32_t tx_buffer[500];

uint32_t time_stamp[2];

ErrStatus status;

status = enet_ptpframe_transmit_normal_mode(tx_buffer, 500, time_stamp);

```

enet_wum_filter_register_pointer_reset

The description of enet_wum_filter_register_pointer_reset is shown as below:

Table 3-292. Function enet_wum_filter_register_pointer_reset

Function name	enet_wum_filter_register_pointer_reset
Function prototype	void enet_wum_filter_register_pointer_reset(void);
Function descriptions	wakeup frame filter register pointer reset
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* reset wakeup frame filter register pointer */

enet_wum_filter_register_pointer_reset ();

```

enet_wum_filter_config

The description of enet_wum_filter_config is shown as below:

Table 3-293. Function enet_wum_filter_config

Function name	enet_wum_filter_config
Function prototype	void enet_wum_filter_config(uint32_t pdata[]);
Function descriptions	set the remote wakeup frame registers
Precondition	-

The called functions		-
Input parameter{in}		
pdata	pointer to buffer data which is written to remote wakeup frame registers (8 words total)	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

```
/* set the remote wakeup frame registers */

uint32_t wum_data[8];

enet_wum_filter_config (wum_data);
```

enet_wum_feature_enable

The description of `enet_wum_feature_enable` is shown as below:

Table 3-294. Function `enet_wum_feature_enable`

Function name	enet_wum_feature_enable
Function prototype	void enet_wum_feature_enable(uint32_t feature);
Function descriptions	enable wakeup management features
Precondition	-
The called functions	-
Input parameter{in}	
feature	one or more parameters can be selected which are shown as below
<i>ENET_WUM_POWER_DOWN</i>	power down mode
<i>ENET_WUM_MAGIC_PACKET_FRAME</i>	enable a wakeup event due to magic packet reception
<i>ENET_WUM_WAKE_UP_FRAME</i>	enable a wakeup event due to wakeup frame reception
<i>ENET_WUM_GLOBAL_UNICAST</i>	any received unicast frame passed filter is considered to be a wakeup frame

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable power down mode */
enet_wum_feature_enable(ENET_WUM_POWER_DOWN);
```

enet_wum_feature_disable

The description of enet_wum_feature_disable is shown as below:

Table 3-295. Function enet_wum_feature_disable

Function name	enet_wum_feature_disable
Function prototype	void enet_wum_feature_disable(uint32_t feature)
Function descriptions	disable wakeup management features
Precondition	-
The called functions	-
Input parameter{in}	
feature	one or more parameters can be selected which are shown as below
<i>ENET_WUM_MAGIC_PACKET_FRAME</i>	enable a wakeup event due to magic packet reception
<i>ENET_WUM_WAKE_UP_FRAME</i>	enable a wakeup event due to wakeup frame reception
<i>ENET_WUM_GLOBAL_UNICAST</i>	any received unicast frame passed filter is considered to be a wakeup frame
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable power down mode */
```

```
enet_wum_feature_disable(ENET_WUM_POWER_DOWN);
```

enet_msc_counters_reset

The description of enet_msc_counters_reset is shown as below:

Table 3-296. Function enet_msc_counters_reset

Function name	enet_msc_counters_reset
Function prototype	void enet_msc_counters_reset(void);
Function descriptions	reset the MAC statistics counters
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the MAC statistics counters */
enet_msc_counters_reset();
```

enet_msc_feature_enable

The description of enet_msc_feature_enable is shown as below:

Table 3-297. Function enet_msc_feature_enable

Function name	enet_msc_feature_enable
Function prototype	void enet_msc_feature_enable(uint32_t feature);
Function descriptions	enable the MAC statistics counter features
Precondition	-
The called functions	-
Input parameter{in}	
feature	one or more parameters can be selected which are shown as below

<i>ENET_MSC_COUNTE R_STOP_ROLLOVER</i>	counter stop rollover
<i>ENET_MSC_RESET_O N_READ</i>	reset on read
<i>ENET_MSC_COUNTE RS_FREEZE</i>	MSC counter freeze
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable counter stop rollover function */
enet_msc_feature_enable(ENET_MSC_COUNTER_STOP_ROLLOVER);
```

enet_msc_feature_disable

The description of enet_msc_feature_disable is shown as below:

Table 3-298. Function enet_msc_feature_disable

Function name	enet_msc_feature_disable
Function prototype	void enet_msc_feature_disable(uint32_t feature);
Function descriptions	disable the MAC statistics counter features
Precondition	-
The called functions	-
Input parameter{in}	
feature	one or more parameters can be selected which are shown as below
<i>ENET_MSC_COUNTE R_STOP_ROLLOVER</i>	counter stop rollover
<i>ENET_MSC_RESET_O N_READ</i>	reset on read
<i>ENET_MSC_COUNTE RS_FREEZE</i>	MSC counter freeze
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable counter stop rollover function */
enet_msc_feature_disable(ENET_MSC_COUNTER_STOP_ROLLOVER);
```

enet_msc_counters_preset_config

The description of `enet_msc_counters_preset_config` is shown as below:

Table 3-299. Function `enet_msc_counters_preset_config`

Function name	enet_msc_counters_preset_config
Function prototype	void enet_msc_counters_preset_config(enet_msc_preset_enum mode);
Function descriptions	configure MAC statistics counters preset mode
Precondition	-
The called functions	-
Input parameter{in}	
mode	MSC counters preset mode, refer to <code>enet_msc_preset_enum</code> only one parameter can be selected which is shown as below
<code>ENET_MSC_PRESET_NONE</code>	do not preset MSC counter
<code>ENET_MSC_PRESET_HALF</code>	preset all MSC counters to almost-half(0x7FFF FFF0) value
<code>ENET_MSC_PRESET_FULL</code>	preset all MSC counters to almost-full(0xFFFF FFF0) value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* preset all MSC counters to almost-half */
```

enet_msc_counters_preset_config (ENET_MSC_PRESET_HALF);

enet_msc_counters_get

The description of enet_msc_counters_get is shown as below:

Table 3-300. Function enet_msc_counters_get

Function name	enet_msc_counters_get
Function prototype	uint32_t enet_msc_counters_get(enet_msc_counter_enum counter);
Function descriptions	get MAC statistics counter
Precondition	-
The called functions	-
Input parameter{in}	
counter	MSC counters which is selected only one parameter can be selected which is shown as below
<i>ENET_MSC_TX_SCCN T</i>	MSC transmitted good frames after a single collision counter
<i>ENET_MSC_TX_MSC CNT</i>	MSC transmitted good frames after more than a single collision counter
<i>ENET_MSC_TX_TGFC NT</i>	MSC transmitted good frames counter
<i>ENET_MSC_RX_RFCE CNT</i>	MSC received frames with CRC error counter
<i>ENET_MSC_RX_RFAE CNT</i>	MSC received frames with alignment error counter
<i>ENET_MSC_RX_RGU FCNT</i>	MSC received good unicast frames counter
Output parameter{out}	
-	-
Return value	
uint32_t	the MSC counter value

Example:

```
/* get MSC transmitted good frames after a single collision counter value*/
```

```
uin32_t reval;
```

```
reval = enet_msc_counters_get(ENET_MSC_TX_SCCNT);
```

enet_ptp_feature_enable

The description of `enet_ptp_feature_enable` is shown as below:

Table 3-301. Function `enet_ptp_feature_enable`

Function name	enet_ptp_feature_enable
Function prototype	void enet_ptp_feature_enable(uint32_t feature);
Function descriptions	enable the PTP features
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET PTP mode one or more parameters can be selected which are shown as below
<i>ENET_RXTX_TIMESTAMP</i> <i>AMP</i>	timestamp function for transmit and receive frames
<i>ENET_PTP_TIMESTAMP</i> <i>MP_INT</i>	timestamp interrupt trigger
<i>ENET_ALL_RX_TIMESTAMP</i> <i>TAMP</i>	all received frames are taken snapshot
<i>ENET_NONTYPE_FRAME_SNAPSHOT</i>	take snapshot when received non type frame
<i>ENET_IPV6_FRAME_SNAPSHOT</i>	take snapshot for IPv6 frame
<i>ENET_IPV4_FRAME_SNAPSHOT</i>	take snapshot for IPv4 frame
<i>ENET_PTP_FRAME_USE_MACADDRESS_FILTER</i>	use MAC address1-3 to filter the PTP frame
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable PTP function for all received frames */

enet_ptp_feature_enable(ENET_ALL_RX_TIMESTAMP);

```

enet_ptp_feature_disable

The description of enet_ptp_feature_disable is shown as below:

Table 3-302. Function enet_ptp_feature_disable

Function name	enet_ptp_feature_disable
Function prototype	void enet_ptp_feature_disable(uint32_t feature);
Function descriptions	disable the PTP features
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET PTP mode one or more parameters can be selected which are shown as below
<i>ENET_RXTX_TIMESTAMP</i>	timestamp function for transmit and receive frames
<i>ENET_PTP_TIMESTAMP_MP_INT</i>	timestamp interrupt trigger
<i>ENET_ALL_RX_TIMESTAMP</i>	all received frames are taken snapshot
<i>ENET_NONTYPE_FRAME_SNAPSHOT</i>	take snapshot when received non type frame
<i>ENET_IPV6_FRAME_SNAPSHOT</i>	take snapshot for IPv6 frame
<i>ENET_IPV4_FRAME_SNAPSHOT</i>	take snapshot for IPv4 frame
<i>ENET_PTP_FRAME_USE_MACADDRESS_FILTER</i>	use MAC address1-3 to filter the PTP frame
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable PTP function for all received frames */

enet_ptp_feature_disable(ENET_ALL_RX_TIMESTAMP);
```

enet_ptp_timestamp_function_config

The description of enet_ptp_timestamp_function_config is shown as below:

Table 3-303. Function enet_ptp_timestamp_function_config

Function name	enet_ptp_timestamp_function_config
Function prototype	ErrStatus enet_ptp_timestamp_function_config(enet_ptp_function_enum func);
Function descriptions	configure the PTP timestamp function
Precondition	-
The called functions	-
Input parameter{in}	
func	only one parameter can be selected which is shown as below
ENET_CKNT_ORDINARY	type of ordinary clock node type for timestamp
ENET_CKNT_BOUNDARY	type of boundary clock node type for timestamp
ENET_CKNT_END_TO_END	type of end-to-end transparent clock node type for timestamp
ENET_CKNT_PEER_TO_PEER	type of peer-to-peer transparent clock node type for timestamp
ENET_PTP_ADDEND_UPDATE	addend register update
ENET_PTP_SYSTIME_UPDATE	timestamp update
ENET_PTP_SYSTIME_INIT	timestamp initialize
ENET_PTP_FINEMOD	the system timestamp uses the fine method for updating

<i>E</i>	
<i>ENET_PTP_COARSE_MODE</i>	the system timestamp uses the coarse method for updating
<i>ENET_SUBSECOND_DIGITAL_ROLLOVER</i>	digital rollover mode
<i>ENET_SUBSECOND_BINARY_ROLLOVER</i>	digital rollover mode
<i>ENET_SNOOPING_PT_P_VERSION_2</i>	version 2
<i>ENET_SNOOPING_PT_P_VERSION_1</i>	version 1
<i>ENET_EVENT_TYPE_MESSAGES_SNAPSHOT</i>	only event type messages are taken snapshot
<i>ENET_ALL_TYPE_MESSAGES_SNAPSHOT</i>	all type messages are taken snapshot except announce, management and signaling message
<i>ENET_MASTER_NODE_MESSAGE_SNAPS_HOT</i>	snapshot is only take for master node message
<i>ENET_SLAVE_NODE_MESSAGE_SNAPSHOT</i>	snapshot is only taken for slave node message
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```

/* config addend register update function */

ErrStatus eval = ERROR;

eval = enet_ptp_timestamp_function_config(ENET_PTP_ADDEND_UPDATE);

```

enet_ptp_subsecond_increment_config

The description of `enet_ptp_subsecond_increment_config` is shown as below:

Table 3-304. Function enet_ptp_subsecond_increment_config

Function name	enet_ptp_subsecond_increment_config
Function prototype	void enet_ptp_subsecond_increment_config(uint32_t subsecond);
Function descriptions	configure system time subsecond increment value
Precondition	-
The called functions	-
Input parameter{in}	
subsecond	the value will be added to the subsecond value of system time, this value must be between 0 and 0xFF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure 0x1F as system time subsecond increment value */
enet_ptp_subsecond_increment_config(0x1F);
```

enet_ptp_timestamp_addend_config

The description of enet_ptp_timestamp_addend_config is shown as below:

Table 3-305. Function enet_ptp_timestamp_addend_config

Function name	enet_ptp_timestamp_addend_config
Function prototype	void enet_ptp_timestamp_addend_config(uint32_t add);
Function descriptions	adjusting the clock frequency only in fine update mode
Precondition	-
The called functions	-
Input parameter{in}	
add	the value will be added to the accumulator register to achieve time synchronization (0 – 0xFFFF FFFF)
Output parameter{out}	
-	-

Return value	
-	-

/* added 0x1FFF to the accumulator register */

```
enet_ptp_timestamp_addend_config(0x1FFF);
```

enet_ptp_timestamp_update_config

The description of enet_ptp_timestamp_update_config is shown as below:

Table 3-306. Function enet_ptp_timestamp_update_config

Function name	enet_ptp_timestamp_update_config
Function prototype	void enet_ptp_timestamp_update_config(uint32_t sign, uint32_t second, uint32_t subsecond);
Function descriptions	initialize or add/subtract to second of the system time
Precondition	-
The called functions	-
Input parameter{in}	
sign	timestamp update positive or negative sign, only one parameter can be selected which is shown as below
<i>ENET_PTP_ADD_TO_TIME</i>	update value is added to system time
<i>ENET_PTP_SUBSTRACT_FROM_TIME</i>	timestamp update value is subtracted from system time
Input parameter{in}	
second	initializing or adding/subtracting to second of the system time (0 – 0xFFFF FFFF)
Input parameter{in}	
subsecond	the current subsecond of the system time with 0.46 ns accuracy (0 – 0x7FFF FFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize system time with timestamp update value */

enet_ptp_timestamp_update_config(ENET_PTP_ADD_TO_TIME, 0, 0);

```

enet_ptp_expected_time_config

The description of enet_ptp_expected_time_config is shown as below:

Table 3-307. Function enet_ptp_expected_time_config

Function name	enet_ptp_expected_time_config
Function prototype	void enet_ptp_expected_time_config(uint32_t second, uint32_t nanosecond);
Function descriptions	configure the expected target time
Precondition	-
The called functions	-
Input parameter{in}	
second	the expected target second time (0 – 0xFFFF FFFF)
Input parameter{in}	
nanosecond	the expected target nanosecond time (signed) (0 – 0xFFFF FFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure the expected target time */

enet_ptp_expected_time_config(2000, 0);

```

enet_ptp_system_time_get

The description of enet_ptp_system_time_get is shown as below:

Table 3-308. Function enet_ptp_system_time_get

Function name	enet_ptp_system_time_get
Function prototype	void enet_ptp_system_time_get(enet_ptp_systime_struct *systime_struct);

Function descriptions	get the current system time
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
systime_struct	pointer to a enet_ptp_systime_struct structure which contains parameters of PTP system time, the structure members can refer to <u>Structure</u> <u>enet_ptp_systime_struct</u>
Return value	
-	-

Example:

```

/* get the current system time */

enet_ptp_systime_struct systime;

enet_ptp_system_time_get(&systime);

enet_ptp_pps_output_frequency_config

```

The description of enet_ptp_pps_output_frequency_config is shown as below:

Table 3-309. Function enet_ptp_pps_output_frequency_config

Function name	enet_ptp_pps_output_frequency_config
Function prototype	void enet_ptp_pps_output_frequency_config(uint32_t freq);
Function descriptions	configure the PPS output frequency
Precondition	-
The called functions	-
Input parameter{in}	
freq	
<i>ENET_PPSOFC_1HZ</i>	PPS output 1Hz frequency
<i>ENET_PPSOFC_2HZ</i>	PPS output 2Hz frequency
<i>ENET_PPSOFC_4HZ</i>	PPS output 4Hz frequency
<i>ENET_PPSOFC_8HZ</i>	PPS output 8Hz frequency

<i>ENET_PPSOFC_16HZ</i>	PPS output 16Hz frequency
<i>ENET_PPSOFC_32HZ</i>	PPS output 32Hz frequency
<i>ENET_PPSOFC_64HZ</i>	PPS output 64Hz frequency
<i>ENET_PPSOFC_128HZ</i>	PPS output 128Hz frequency
<i>ENET_PPSOFC_256HZ</i>	PPS output 256Hz frequency
<i>ENET_PPSOFC_512HZ</i>	PPS output 512Hz frequency
<i>ENET_PPSOFC_1024HZ</i>	PPS output 1024Hz frequency
<i>ENET_PPSOFC_2048HZ</i>	PPS output 2048Hz frequency
<i>ENET_PPSOFC_4096HZ</i>	PPS output 4096Hz frequency
<i>ENET_PPSOFC_8192HZ</i>	PPS output 8192Hz frequency
<i>ENET_PPSOFC_16384HZ</i>	PPS output 16384Hz frequency
<i>ENET_PPSOFC_32768HZ</i>	PPS output 32768Hz frequency
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure the PPS output frequency as 1Hz */
enet_ptp_pps_output_frequency_config(ENET_PPSOFC_1HZ);

```

enet_initpara_reset

The description of **enet_initpara_reset** is shown as below:

Table 3-310. Function enet_initpara_reset

Function name	enet_initpara_reset
Function prototype	void enet_initpara_reset(void);
Function descriptions	reset the ENET initpara struct, call it before using enet_initpara_config()
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the ENET initpara struct */
enet_initpara_reset();
```

3.11. EXMC

The external memory controller EXMC, is used as a translator for MCU to access a variety of external memory. The EXMC registers are listed in chapter [3.11.1](#), the EXMC firmware functions are introduced in chapter [3.11.2](#).

3.11.1. Descriptions of Peripheral registers

EXMC registers are listed in the table shown as below:

Table 3-311. EXMC Registers

Registers	Descriptions
EXMC_SNCTL	SRAM/NOR Flash control registers
EXMC_SNTCFG	SRAM/NOR Flash timing configuration registers
EXMC_SNWTCFG	SRAM/NOR Flash write timing configuration registers
EXMC_NPCTL	NAND flash/PC card control registers
EXMC_NPINTEN	NAND flash/PC card interrupt enable registers
EXMC_NPCTCFG	NAND flash/PC card common space timing configuration registers
EXMC_NPATCFG	NAND flash/PC card attribute space timing configuration registers
EXMC_PIOTCFG3	PC card I/O space timing configuration register

Registers	Descriptions
EXMC_NECC	NAND flash ECC registers
EXMC_SDCTL	SDRAM control registers
EXMC_SDTCFG	SDRAM timing configuration registers
EXMC_SDCMD	SDRAM command register
EXMC_SDARI	SDRAM auto-refresh interval register
EXMC_SDSTAT	SDRAM status register
EXMC_SDRSCTL	SDRAM read sample control register
EXMC_SINIT	SPI initialization register
EXMC_SRCMD	SPI read command register
EXMC_SWCMD	SPI write command register
EXMC_SIDL	SPI ID low register
EXMC_SIDH	SPI ID high register

3.11.2. Descriptions of Peripheral functions

EXMC firmware functions are listed in the table shown as below:

Table 3-312. EXMC firmware function

Function name	Function description
exmc_norsram_deinit	deinitialize EXMC NOR/SRAM regionx
exmc_norsram_struct_para_init	initialize exmc_norsram_parameter_struct with the default values
exmc_norsram_init	initialize EXMC NOR/SRAM regionx
exmc_norsram_enable	enable EXMC NOR/PSRAM regionx
exmc_norsram_disable	disable EXMC NOR/PSRAM regionx
exmc_nand_deinit	deinitialize EXMC NAND bankx
exmc_nand_struct_para_init	initialize exmc_norsram_parameter_struct with the default values
exmc_nand_init	initialize EXMC NAND bankx
exmc_nand_enable	enable EXMC NAND bankx
exmc_nand_disable	disable EXMC NAND bankx
exmc_pccard_deinit	deinitialize EXMC PC card bank
exmc_pccard_struct_para_init	initialize exmc_pccard_parameter_struct with the default values
exmc_pccard_init	initialize EXMC PC card bank
exmc_pccard_enable	enable EXMC PC card bank
exmc_pccard_disable	disable EXMC PC card bank
exmc_sdram_deinit	deinitialize EXMC SDRAM device
exmc_sdram_struct_para_init	initialize exmc_sdram_parameter_struct with the default values
exmc_sdram_init	initialize EXMC SDRAM device
exmc_sdram_struct_command_para_i	initialize exmc_sdram_command_parameter_struct with the

Function name	Function description
nit	default values
exmc_sqipsram_deinit	deinitialize EXMC SQPIPSRAM
exmc_sqipsram_struct_para_init	initialize exmc_sqipsram_parameter_struct with the default values
exmc_sqipsram_init	initialize EXMC SQPIPSRAM
exmc_norsram_consecutive_clock_config	configure consecutive clock
exmc_norsram_page_size_config	configure CRAM page size
exmc_nand_ecc_config	enable or disable the EXMC NAND ECC function
exmc_ecc_get	get the EXMC ECC value
exmc_sdram_readsampEnable	enable or disable read sample
exmc_sdram_readsampConfig	configure the delayed sample clock of read data
exmc_sdram_command_config	configure the SDRAM memory command
exmc_sdram_refresh_count_set	set auto-refresh interval
exmc_sdram_autorefresh_number_set	set the number of successive auto-refresh command
exmc_sdram_write_protection_config	config the write protection function
exmc_sdram_bankstatus_get	get the status of SDRAM device0 or device1
exmc_sqipsram_read_command_set	set the read command
exmc_sqipsram_write_command_set	set the write command
exmc_sqipsram_read_id_command_send	send SPI read ID command
exmc_sqipsram_write_cmd_send	send SPI special command which does not have address and data phase
exmc_sqipsram_low_id_get	get the EXMC SPI ID low data
exmc_sqipsram_high_id_get	get the EXMC SPI ID high data
exmc_sqipsram_send_command_status_get	get the bit value of EXMC send write command bit or read ID command
exmc_interrupt_enable	enable EXMC interrupt
exmc_interrupt_disable	disable EXMC interrupt
exmc_flag_get	get EXMC flag status
exmc_flag_clear	clear EXMC flag status
exmc_interrupt_flag_get	get EXMC interrupt flag
exmc_interrupt_flag_clear	clear EXMC interrupt flag

Structure exmc_norsram_timing_parameter_struct

Table 3-313. Structure exmc_norsram_timing_parameter_struct

Member name	Function description
asyn_access_mode	asynchronous access mode
syn_data_latency	configure the data latency

syn_clk_division	configure the clock divide ratio
bus_latency	configure the bus latency
asyn_data_setuptime	configure the data setup time, asynchronous access mode valid
asyn_address_holdtime	configure the address hold time, asynchronous access mode valid
asyn_address_setuptime	configure the data setup time, asynchronous access mode valid

Structure exmc_norsram_parameter_struct

Table 3-314. Structure exmc_norsram_parameter_struct

Member name	Function description
norsram_region	select the region of EXMC NOR/SRAM
write_mode	the write mode, synchronous mode or asynchronous mode
extended_mode	enable or disable the extended mode
asyn_wait	enable or disable the asynchronous wait function
nwait_signal	enable or disable the NWAIT signal while in synchronous burst mode
memory_write	enable or disable the write operation
nwait_config	NWAIT signal configuration
wrap_burst_mode	enable or disable the wrap burst mode
nwait_polarity	specifies the polarity of NWAIT signal from memory
burst_mode	enable or disable the burst mode
databus_width	specifies the databus width of external memory
memory_type	specifies the type of external memory
address_data_mux	specifies whether the data bus and address bus are multiplexed
read_write_timing	timing parameters for read and write if the extended mode is not used or the timing parameters for read if the extended mode is used
write_timing	timing parameters for write when the extended mode is used

Structure exmc_nand_pccard_timing_parameter_struct

Table 3-315. Structure exmc_nand_pccard_timing_parameter_struct

Member name	Function description
databus_hiztime	configure the databus HiZ time for write operation
holdtime	configure the address hold time(or the data hold time for write operation)
waittime	configure the minimum wait time
setuptime	configure the address setup time

Structure exmc_nand_parameter_struct

Table 3-316. Structure exmc_nand_parameter_struct

Member name	Function description
nand_bank	select the bank of NAND

Member name	Function description
ecc_size	the page size for the ECC calculation
atr_latency	configure the latency of ALE low to RB low
ctr_latency	configure the latency of CLE low to RB low
ecc_logic	enable or disable the ECC calculation logic
databus_width	the NAND flash databus width
wait_feature	enable or disable the wait feature
common_space_timing	the timing parameters for NAND flash common space
attribute_space_timing	the timing parameters for NAND flash attribute space

Structure exmc_pccard_parameter_struct

Table 3-317. Structure exmc_pccard_parameter_struct

Member name	Function description
atr_latency	configure the latency of ALE low to RB low
ctr_latency	configure the latency of CLE low to RB low
wait_feature	enable or disable the wait feature
common_space_timing	the timing parameters for PC card common space
attribute_space_timing	the timing parameters for PC card attribute space
io_space_timing	the timing parameters for PC card IO space

Structure exmc_sdram_timing_parameter_struct

Table 3-318. Structure exmc_sdram_timing_parameter_struct

Member name	Function description
row_to_column_delay	configure the row to column delay
row_prelude_delay	configure the row precharge delay
write_recovery_delay	configure the write recovery delay
auto_refresh_delay	configure the auto refresh delay
row_address_select_delay	configure the row address select delay
exit_selfrefresh_delay	configure the exit self-refresh delay
load_mode_register_delay	configure the load mode register delay

Structure exmc_sdram_parameter_struct

Table 3-319. Structure exmc_sdram_parameter_struct

Member name	Function description
sdram_device	select the device of SDRAM
pipeline_read_delay	the delay for reading data after CAS latency in HCLK clock cycles
burst_read_switch	enable or disable the burst read
sdclock_config	the SDCLK memory clock for both SDRAM banks
write_protection	enable or disable SDRAM bank write protection function
cas_latency	configure the SDRAM CAS latency
internal_bank_number	the number of internal bank
data_width	the databus width of SDRAM memory
row_address_width	the bit width of a row address
column_address_width	the bit width of a column address
timing	the timing parameters for write and read SDRAM

Structure exmc_sdram_command_parameter_struct

Table 3-320. Structure exmc_sdram_command_parameter_struct

Member name	Function description
mode_register_content	the SDRAM mode register content
auto_refresh_number	the number of successive auto-refresh cycles will be send when CMD = 011
bank_select	the bank which command will be sent to
command	the commands that will be sent to SDRAM

Structure exmc_sqipsram_parameter_struct

Table 3-321. Structure exmc_sqipsram_parameter_struct

Member name	Function description
sample_polarity	read data sample polarity
id_length	SPI PSRAM ID length
address_bits	bit number of SPI PSRAM address phase
command_bits	bit number of SPI PSRAM command phase

exmc_norsram_deinit

The description of exmc_norsram_deinit is shown as below:

Table 3-322. Function exmc_norsram_deinit

Function name	
	exmc_norsram_deinit

Function prototype	void exmc_norsram_deinit(uint32_t exmc_norsram_region);
Function descriptions	deinitialize EXMC NOR/SRAM region
Precondition	-
The called functions	-
Input parameter{in}	
exmc_norsram_region n	EXMC NOR/SRAM region
EXMC_BANK0_NORS RAM_REGIONx	x=0,1,2,3
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* deinitialize the EXMC NOR/SRAM region1 of bank0 */

exmc_norsram_deinit(EXMC_BANK0_NORSRAM_REGION1);

```

exmc_norsram_struct_para_init

The description of exmc_norsram_struct_para_init is shown as below:

Table 3-323. Function exmc_norsram_struct_para_init

Function name	exmc_norsram_struct_para_init
Function prototype	void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
Function descriptions	initialize the struct exmc_norsram_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
exmc_norsram_init_st ruct	Structure for initialization, the structure members can refer to Table 3-314. <u>Structure exmc_norsram_parameter_struct</u>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize the struct nor_init_struct */

exmc_norsram_parameter_struct nor_init_struct;
exmc_norsram_struct_para_init (&nor_init_struct);

```

exmc_norsram_init

The description of exmc_norsram_init is shown as below:

Table 3-324. Function exmc_norsram_init

Function name	exmc_norsram_init
Function prototype	void exmc_norsram_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
Function descriptions	initialize EXMC NOR/SRAM region
Precondition	-
The called functions	-
Input parameter{in}	
exmc_norsram_init_st ruct	Structure for initialization, the structure members can refer to Table 3-314. Structure exmc_norsram_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize EXMC NOR/SRAM bank */

exmc_norsram_parameter_struct lcd_init_struct;
exmc_norsram_timing_parameter_struct lcd_timing_init_struct;

/* configure timing parameter */

lcd_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;
lcd_timing_init_struct.syn_data_latency = EXMC_DATALAT_2_CLK;
lcd_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;
lcd_timing_init_struct.bus_latency = 1;
lcd_timing_init_struct.asyn_data_setuptime = 5;
lcd_timing_init_struct.asyn_address_holdtime = 2;
lcd_timing_init_struct.asyn_address_setuptime = 2;

/* configure EXMC bus parameters */

lcd_init_struct.norsram_region = EXMC_BANK0_NORSRAM_REGION1;
lcd_init_struct.write_mode = EXMC_ASYN_WRITE;
lcd_init_struct.extended_mode = DISABLE;
lcd_init_struct.asyn_wait = DISABLE;

```

```

lcd_init_struct.nwait_signal = DISABLE;
lcd_init_struct.memory_write = ENABLE;
lcd_init_struct.nwait_config = EXMC_NWAIT_CONFIG_BEFORE;
lcd_init_struct.wrap_burst_mode = DISABLE;
lcd_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;
lcd_init_struct.burst_mode = DISABLE;
lcd_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;
lcd_init_struct.memory_type = EXMC_MEMORY_TYPE_SRAM;
lcd_init_struct.address_data_mux = DISABLE;
lcd_init_struct.read_write_timing = &lcd_timing_init_struct;
lcd_init_struct.write_timing = &lcd_timing_init_struct;
exmc_norsram_init(&lcd_init_struct);

```

exmc_norsram_enable

The description of exmc_norsram_enable is shown as below:

Table 3-325. Function exmc_norsram_enable

Function name	exmc_norsram_enable
Function prototype	void exmc_norsram_enable(uint32_t exmc_norsram_region);
Function descriptions	enable EXMC NOR/PSRAM region
Precondition	-
The called functions	-
Input parameter{in}	
exmc_norsram_region n	EXMC NOR/SRAM region
EXMC_BANK0_NORS RAM_REG/ONx	x=0,1,2,3
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable the EXMC NOR/SRAM region1 of bank0 */
exmc_norsram_enable(EXMC_BANK0_NORSRAM_REGION1);

```

exmc_norsram_disable

The description of exmc_norsram_disable is shown as below:

Table 3-326. Function exmc_norsram_disable

Function name	exmc_norsram_disable
Function prototype	void exmc_norsram_disable(uint32_t exmc_norsram_region);
Function descriptions	disable EXMC NOR/PSRAM region
Precondition	-
The called functions	-
Input parameter{in}	
exmc_norsram_region n	EXMC NOR/SRAM region
EXMC_BANK0_NORS RAM_REGIONx	x=0,1,2,3
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the EXMC NOR/SRAM region1 of bank0 */
exmc_norsram_disable(EXMC_BANK0_NORSRAM_REGION1);
```

exmc_nand_deinit

The description of exmc_nand_deinit is shown as below:

Table 3-327. Function exmc_nand_deinit

Function name	exmc_nand_deinit
Function prototype	void exmc_nand_deinit(uint32_t exmc_nand_bank);
Function descriptions	deinitialize EXMC NAND bank
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_bank	the bank of NAND
EXMC_BANKx_NAND	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize EXMC NOR/SRAM bank1 */
```

```
exmc_norsram_deinit(EXMC_BANK1_NAND);
```

exmc_nand_struct_para_init

The description of exmc_nand_struct_para_init is shown as below:

Table 3-328. Function exmc_nand_struct_para_init

Function name	exmc_nand_struct_para_init
Function prototype	void exmc_nand_struct_para_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
Function descriptions	initialize the struct exmc_nand_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_init_struct t	Structure for initialization, the structure members can refer to Table 3-316. <u>Structure exmc_nand_parameter_struct</u>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the struct nand_init_struct */

exmc_nand_parameter_struct nand_init_struct;
exmc_nand_struct_para_init (&nand_init_struct);
```

exmc_nand_init

The description of exmc_nand_init is shown as below:

Table 3-329. Function exmc_nand_init

Function name	exmc_nand_init
Function prototype	void exmc_nand_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
Function descriptions	initialize EXMC NAND bank
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_init_struct t	Structure for initialization, the structure members can refer to Table 3-316. <u>Structure exmc_nand_parameter_struct</u>
Output parameter{out}	
-	-
Return value	

Example:

```

exmc_nand_parameter_struct nand_init_struct;

exmc_nand_pccard_timing_parameter_struct nand_timing_init_struct;

/* EXMC configuration */

nand_timing_init_struct.setuptime = 5;

nand_timing_init_struct.waittime = 4;

nand_timing_init_struct.holdtime = 2;

nand_timing_init_struct.databus_hiztime = 2;

nand_init_struct.nand_bank = EXMC_BANK1_NAND;

nand_init_struct.ecc_size = EXMC_ECC_SIZE_2048BYTES;

nand_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;

nand_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;

nand_init_struct.ecc_logic = ENABLE;

nand_init_struct.databus_width = EXMC_NAND_DATABUS_WIDTH_8B;

nand_init_struct.wait_feature = ENABLE;

nand_init_struct.common_space_timing = &nand_timing_init_struct;

nand_init_struct.attribute_space_timing = &nand_timing_init_struct;

exmc_nand_init(&nand_init_struct);

```

exmc_nand_enable

The description of exmc_nand_enable is shown as below:

Table 3-330. Function exmc_nand_enable

Function name	exmc_nand_enable
Function prototype	void exmc_nand_enable(uint32_t exmc_nand_bank);
Function descriptions	enable EXMC NAND bank
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_bank	the bank of NAND
EXMC_BANKx_NAND	x=1,2
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* enable EXMC NAND bank1 */

exmc_nand_enable(EXMC_BANK1_NAND);
```

exmc_nand_disable

The description of exmc_nand_disable is shown as below:

Table 3-331. Function exmc_nand_disable

Function name	exmc_nand_disable
Function prototype	exmc_nand_disable(uint32_t exmc_nand_bank);
Function descriptions	disable EXMC NAND bank
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_bank	the bank of NAND
<i>EXMC_BANKx_NAND</i>	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable EXMC NAND bank1 */

exmc_nand_disable(EXMC_BANK1_NAND);
```

exmc_pccard_deinit

The description of exmc_pccard_deinit is shown as below:

Table 3-332. Function exmc_pccard_deinit

Function name	exmc_pccard_deinit
Function prototype	void exmc_pccard_deinit(void);
Function descriptions	deinitialize EXMC PC card bank
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* deinitialize EXMC PC card bank */

exmc_pccard_deinit();
```

exmc_pccard_struct_para_init

The description of exmc_pccard_struct_para_init is shown as below:

Table 3-333. Function exmc_pccard_struct_para_init

Function name	exmc_pccard_struct_para_init
Function prototype	void exmc_pccard_struct_para_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);
Function descriptions	initialize the struct exmc_pccard_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
exmc_pccard_init_struct	Structure for initialization, the structure members can refer to Table 3-317. <u>Structure exmc_pccard_parameter_struct</u>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the struct pccard_init_struct */

exmc_pccard_parameter_struct pccard_init_struct;
exmc_pccard_struct_para_init (&pccard_init_struct);
```

exmc_pccard_init

The description of exmc_pccard_init is shown as below:

Table 3-334. Function exmc_pccard_init

Function name	exmc_pccard_init
Function prototype	void exmc_pccard_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);
Function descriptions	initialize EXMC PC card bank
Precondition	-
The called functions	-
Input parameter{in}	

exmc_pccard_init_struct	Structure for initialization, the structure members can refer to Table 3-317. Structure exmc_pccard_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

exmc_pccard_parameter_struct pccard_init_struct;
exmc_nand_pccard_timing_parameter_struct pccard_timing_init_struct;

/* EXMC configuration */

pccard_timing_init_struct.setuptime = 5;
pccard_timing_init_struct.waittime = 4;
pccard_timing_init_struct.holdtime = 2;
pccard_timing_init_struct.databus_hiztime = 2;
pccard_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;
pccard_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;
pccard_init_struct.wait_feature = ENABLE;
pccard_init_struct.common_space_timing = & pccard_timing_init_struct;
pccard_init_struct.attribute_space_timing = & pccard_timing_init_struct;
pccard_init_struct.io_space_timing = & pccard_timing_init_struct;
exmc_pccard_init(&pccard_init_struct);

```

exmc_pccard_enable

The description of `exmc_pccard_enable` is shown as below:

Table 3-335. Function exmc_pccard_enable

Function name	exmc_pccard_enable
Function prototype	void exmc_pccard_enable(void);
Function descriptions	enable EXMC PC card bank
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* enable EXMC PC card bank */
exmc_pccard_enable();
```

exmc_pccard_disable

The description of exmc_pccard_disable is shown as below:

Table 3-336. Function exmc_pccard_disable

Function name	exmc_pccard_disable
Function prototype	void exmc_pccard_disable(void);
Function descriptions	disable EXMC PC card bank
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable EXMC PC card bank */
exmc_pccard_disable();
```

exmc_sdram_deinit

The description of exmc_sdram_deinit is shown as below:

Table 3-337. Function exmc_sdram_deinit

Function name	exmc_sdram_deinit
Function prototype	void exmc_sdram_deinit(uint32_t exmc_sdram_device);
Function descriptions	deinitialize EXMC SDRAM device
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sdram_device	EXMC SDRAM device
EXMC_SDRAM_DEVICE <i>Ex</i>	x=0,1
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* deinitialize EXMC SDRAM device1 */

exmc_sdram_deinit(EXMC_SDRAM_DEVICE1);
```

exmc_sdram_struct_para_init

The description of exmc_sdram_struct_para_init is shown as below:

Table 3-338. Function exmc_sdram_struct_para_init

Function name	exmc_sdram_struct_para_init
Function prototype	exmc_sdram_struct_para_init(exmc_sdram_parameter_struct* exmc_sdram_init_struct);
Function descriptions	initialize the struct exmc_sdram_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sdram_init_struct	Structure for initialization, the structure members can refer to Table 3-319. Structure exmc_sdram_parameter_struct
ct	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the struct sdram_init_struct */

exmc_sdram_parameter_struct sdram_init_struct;
exmc_sdram_struct_para_init (&sdram_init_struct);
```

exmc_sdram_init

The description of exmc_sdram_init is shown as below:

Table 3-339. Function exmc_sdram_init

Function name	exmc_sdram_init
Function prototype	void exmc_sdram_init(exmc_sdram_parameter_struct* exmc_sdram_init_struct);
Function descriptions	initialize EXMC SDRAM device
Precondition	-
The called functions	-

Input parameter{in}	
exmc_sdrdram_init_struct	Structure for initialization, the structure members can refer to Table 3-319. ct
Structure exmc_sdrdram_parameter_struct	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

exmc_sdrdram_parameter_struct      sdrdram_init_struct;
exmc_sdrdram_timing_parameter_struct sdrdram_timing_init_struct;

/* EXMC configuration */

sdrdram_timing_init_struct.load_mode_register_delay = 2;

/* XSRD: min = 67ns */

sdrdram_timing_init_struct.exit_selfrefresh_delay = 7;

/* RASD: min=42ns , max=120k (ns) */

sdrdram_timing_init_struct.row_address_select_delay = 5;

/* ARFD: min=60ns */

sdrdram_timing_init_struct.auto_refresh_delay = 6;

/* WRD: min=1 Clock cycles +6ns */

sdrdram_timing_init_struct.write_recovery_delay = 2;

/* RPD: min=18ns */

sdrdram_timing_init_struct.row_precharge_delay = 2;

/* RCD: min=18ns */

sdrdram_timing_init_struct.row_to_column_delay = 2;

sdrdram_init_struct.sdrdram_device = sdrdram_device;

sdrdram_init_struct.column_address_width = EXMC_SDRAM_COW_ADDRESS_9;

sdrdram_init_struct.row_address_width = EXMC_SDRAM_ROW_ADDRESS_13;

sdrdram_init_struct.data_width = EXMC_SDRAM_DATABUS_WIDTH_16B;

sdrdram_init_struct.internal_bank_number = EXMC_SDRAM_4_INTER_BANK;

sdrdram_init_struct.cas_latency = EXMC_CAS_LATENCY_3_SDCLK;

sdrdram_init_struct.write_protection = DISABLE;

```

```
sDRAM_init_struct.sdclock_config = EXMC_SDCLK_PERIODS_2_HCLK;
sDRAM_init_struct.brust_read_switch = ENABLE;
sDRAM_init_struct.pipeline_read_delay = EXMC_PIPELINE_DELAY_1_HCLK;
sDRAM_init_struct.timing = &sDRAM_timing_init_struct;
/* EXMC SDRAM bank initialization */
exmc_sDRAM_init(&sDRAM_init_struct);
```

exmc_sDRAM_struct_command_para_init

The description of exmc_sDRAM_struct_command_para_init is shown as below:

Table 3-340. Function exmc_sDRAM_struct_command_para_init

Function name	exmc_sDRAM_struct_command_para_init	
Function prototype	void exmc_sDRAM_struct_command_para_init(exmc_sDRAM_command_parameter_struct *exmc_sDRAM_command_init_struct);	
Function descriptions	initialize exmc_sDRAM_command_parameter_struct with the default values	
Precondition	-	
The called functions	-	
Input parameter{in}		
-	-	
Output parameter{out}		
exmc_sDRAM_command_parameter_struct	structure for initialization, the structure members can refer to Table 3-320. Structure exmc_sDRAM_command_parameter_struct	
Return value		
-	-	

Example:

```
/* initialize the structure exmc_sDRAM_command_init_struct */
exmc_sDRAM_command_parameter_struct t exmc_sDRAM_command_init_struct;
exmc_sDRAM_struct_command_para_init (&exmc_sDRAM_command_init_struct);
```

exmc_sqipSRAM_deinit

The description of exmc_sqipSRAM_deinit is shown as below:

Table 3-341. Function exmc_sqipSRAM_deinit

Function name	exmc_sqipSRAM_deinit	
Function prototype	void exmc_sqipSRAM_deinit(void);	
Function descriptions	deinitialize EXMC SQPIPSRAM	
Precondition	-	

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize EXMC SQPIPSRAM */
```

```
exmc_sqipsram_deinit(void);
```

exmc_sqipsram_struct_para_init

The description of exmc_sqipsram_struct_para_init is shown as below:

Table 3-342. Function exmc_sqipsram_struct_para_init

Function name	exmc_sqipsram_struct_para_init
Function prototype	void exmc_sqipsram_struct_para_init(exmc_sqipsram_parameter_struct* exmc_sqipsram_init_struct);
Function descriptions	initialize the struct exmc_sqipsram_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sqipsram_init_struct	Structure for initialization, the structure members can refer to Table 3-321. Structure exmc_sqipsram_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the struct sqipsram_init_struct */
```

```
exmc_sqipsram_parameter_struct sqipsram_init_struct;
```

```
exmc_sqipsram_struct_para_init (&sqipsram_init_struct);
```

exmc_sqipsram_init

The description of exmc_sqipsram_init is shown as below:

Table 3-343. Function exmc_sqipsram_init

Function name	exmc_sqipsram_init
Function prototype	void exmc_sqipsram_init(exmc_sqipsram_parameter_struct*

	exmc_sqipsram_init_struct);
Function descriptions	initialize EXMC SQPIPSRAM
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sqipsram_init_struct	Structure for initialization, the structure members can refer to Table 3-321. <i>Structure exmc_sqipsram_parameter_struct</i>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

exmc_sqipsram_parameter_struct           sqipsram_init_struct;
/* EXMC configuration */
sqipsram_init_struct.sample_polarity = EXMC_SQPIPSRAM_SAMPLE_RISING_EDGE;
sqipsram_init_struct.id_length = EXMC_SQPIPSRAM_ID_LENGTH_64B;
sqipsram_init_struct.address_bits = EXMC_SQPIPSRAM_ADDR_LENGTH_24B;
sqipsram_init_struct.command_bits = EXMC_SQPIPSRAM_COMMAND_LENGTH_8B;
/* EXMC SDRAM bank initialization */
exmc_sqipsram_init(&sqipsram_init_struct);

```

exmc_norsram_consecutive_clock_config

The description of `exmc_norsram_consecutive_clock_config` is shown as below:

Table 3-344. Function `exmc_norsram_consecutive_clock_config`

Function name	exmc_norsram_consecutive_clock_config
Function prototype	void exmc_norsram_consecutive_clock_config(uint32_t clock_mode);
Function descriptions	configure consecutive clock
Precondition	-
The called functions	-
Input parameter{in}	
clock_mode	the mode of consecutive clock
EXMC_CLOCK_SYN_MODE	the clock is generated only during synchronous access
EXMC_CLOCK_UNCONDITIONALLY	the clock is generated unconditionally
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure consecutive clock */

exmc_norsram_consecutive_clock_config(EXMC_CLOCK_SYN_MODE);
```

exmc_norsram_page_size_config

The description of exmc_norsram_page_size_config is shown as below:

Table 3-345. Function exmc_norsram_page_size_config

Function name	exmc_norsram_page_size_config
Function prototype	void exmc_norsram_page_size_config(uint32_t page_size);
Function descriptions	configure CRAM page size
Precondition	-
The called functions	-
Input parameter{in}	
page_size	CRAM page size
<i>EXMC_CRAM_AUTO_SPLIT</i>	the clock is generated only during synchronous access
<i>EXMC_CRAM_PAGE_SIZE_128_BYTES</i>	page size is 128 bytes
<i>EXMC_CRAM_PAGE_SIZE_256_BYTES</i>	page size is 256 bytes
<i>EXMC_CRAM_PAGE_SIZE_512_BYTES</i>	page size is 512 bytes
<i>EXMC_CRAM_PAGE_SIZE_1024_BYTES</i>	page size is 1024 bytes
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CRAM page size */

exmc_norsram_page_size_config (EXMC_CRAM_PAGE_SIZE_128_BYTES);
```

exmc_nand_ecc_config

The description of exmc_nand_ecc_config is shown as below:

Table 3-346. Function exmc_nand_ecc_config

Function name	exmc_nand_ecc_config
Function prototype	void exmc_nand_ecc_config(uint32_t exmc_nand_bank, ControlStatus newvalue);
Function descriptions	enable or disable the EXMC NAND ECC function
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_bank	specifie the NAND bank
EXMC_BANKx_NAND	x=1,2
Input parameter{in}	
newvalue	ENABLE or DISABLE
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the EXMC NAND ECC function */

exmc_nand_ecc_config(EXMC_BANK1_NAND, ENABLE);
```

exmc_ecc_get

The description of exmc_ecc_get is shown as below:

Table 3-347. Function exmc_ecc_get

Function name	exmc_ecc_get
Function prototype	uint32_t exmc_ecc_get(uint32_t exmc_nand_bank);
Function descriptions	get the EXMC ECC value
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_bank	specifie the NAND bank
EXMC_BANKx_NAND	x=1,2
Output parameter{out}	
-	-
Return value	
uint32_t	the error correction code(ECC) value

Example:

```
/* get the EXMC ECC value */

uint32_t ecc_value;
```

```
ecc_value = exmc_ecc_get(EXMC_BANK1_NAND);
```

exmc_sdrsam_readsampEnable

The description of exmc_sdrsam_readsampEnable is shown as below:

Table 3-348. Function exmc_sdrsam_readsampEnable

Function name	exmc_sdrsam_readsampEnable
Function prototype	void exmc_sdrsam_readsampEnable(ControlStatus newvalue);
Function descriptions	enable or disable read sample
Precondition	-
The called functions	-
Input parameter{in}	
newvalue	ENABLE or DISABLE
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable read sample */

exmc_sdrsam_readsampEnable(ENABLE);
```

exmc_sdrsam_readsampConfig

The description of exmc_sdrsam_readsampConfig is shown as below:

Table 3-349. Function exmc_sdrsam_readsampConfig

Function name	exmc_sdrsam_readsampConfig
Function prototype	void exmc_sdrsam_readsampConfig(uint32_t delay_cell, uint32_t extra_hclk);
Function descriptions	configure the delayed sample clock of read data
Precondition	-
The called functions	-
Input parameter{in}	
delay_cell	SDRAM the delayed sample clock of read data
EXMC_SDRAM_x_DEL_AY_CELL	x=0...15
Input parameter{in}	
extra_hclk	sample cycle of read data
EXMC_SDRAM_READSAMPLE_x_EXTRAHC_LK	x=0,1
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* configure the delayed sample clock and sample cycle of read data */

exmc_sdram_readsampel_config(EXMC_SDRAM_1_DELAY_CELL,
EXMC_SDRAM_READSAMPLE_1_EXTRAHCLK);
```

exmc_sdram_command_config

The description of exmc_sdram_command_config is shown as below:

Table 3-350. Function exmc_sdram_command_config

Function name	exmc_sdram_command_config
Function prototype	void exmc_sdram_command_config(exmc_sdram_command_parameter_struct* exmc_sdram_command_init_struct);
Function descriptions	configure the SDRAM memory command
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sdram_comma nd_init_struct	Structure for initialization, the structure members can refer to Table 3-320. Structure exmc_sdram_command_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the SDRAM memory command */

exmc_sdram_command_parameter_struct      sdram_command_init_struct;
sdram_command_init_struct.command = EXMC_SDRAM_CLOCK_ENABLE;
sdram_command_init_struct.bank_select = bank_select;
sdram_command_init_struct.auto_refresh_number      =
EXMC_SDRAM_AUTO_REFRESH_1_SDCLK;
sdram_command_init_struct.mode_register_content = 0;
exmc_sdram_command_config(&sdram_command_init_struct);
```

exmc_sdram_refresh_count_set

The description of exmc_sdram_refresh_count_set is shown as below:

Table 3-351. Function exmc_sdram_refresh_count_set

Function name	exmc_sdram_refresh_count_set
Function prototype	void exmc_sdram_refresh_count_set(uint32_t exmc_count);
Function descriptions	set auto-refresh interval
Precondition	-
The called functions	-
Input parameter{in}	
exmc_count	the number SDRAM clock cycles unit between two successive auto-refresh commands, 0x0000~0xFFFF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the SDRAM auto-refresh rate counter */
exmc_sdram_refresh_count_set(761);
```

exmc_sdram_autorefresh_number_set

The description of exmc_sdram_autorefresh_number_set is shown as below:

Table 3-352. Function exmc_sdram_autorefresh_number_set

Function name	exmc_sdram_autorefresh_number_set
Function prototype	void exmc_sdram_autorefresh_number_set(uint32_t exmc_number);
Function descriptions	set the number of successive auto-refresh command
Precondition	-
The called functions	-
Input parameter{in}	
exmc_number	the number of successive Auto-refresh cycles will be send
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the number of successive auto-refresh command */
exmc_sdram_autorefresh_number_set(10);
```

exmc_sdrdram_write_protection_config

The description of exmc_sdrdram_write_protection_config is shown as below:

Table 3-353. Function exmc_sdrdram_write_protection_config

Function name	exmc_sdrdram_write_protection_config
Function prototype	void exmc_sdrdram_write_protection_config(uint32_t exmc_sdrdram_device, ControlStatus newvalue);
Function descriptions	config the write protection function
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sdrdram_device	specifie the SDRAM device
<i>Ex</i> <i>EXMC_SDRAM_DEVICE</i>	x=0,1
Input parameter{in}	
newvalue	ENABLE or DISABLE
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the write protection function */
exmc_sdrdram_write_protection_config(EXMC_SDRAM_DEVICE1, ENABLE);
```

exmc_sdrdram_bankstatus_get

The description of exmc_sdrdram_bankstatus_get is shown as below:

Table 3-354. Function exmc_sdrdram_bankstatus_get

Function name	exmc_sdrdram_bankstatus_get
Function prototype	uint32_t exmc_sdrdram_bankstatus_get(uint32_t exmc_sdrdram_device);
Function descriptions	get the status of SDRAM device0 or device1
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sdrdram_device	specifie the SDRAM device
<i>Ex</i> <i>EXMC_SDRAM_DEVICE</i>	x=0,1
Output parameter{out}	
-	-
Return value	
uint32_t	the status of SDRAM device

Example:

```

/* get the status of SDRAM device1 */

uint32_t status;

status = exmc_sdrdram_bankstatus_get (EXMC_SDRAM_DEVICE1);

```

exmc_sqipsram_read_command_set

The description of `exmc_sqipsram_read_command_set` is shown as below:

Table 3-355. Function `exmc_sqipsram_read_command_set`

Function name	<code>exmc_sqipsram_read_command_set</code>
Function prototype	<code>void exmc_sqipsram_read_command_set(uint32_t read_command_mode,uint32_t read_wait_cycle,uint32_t read_command_code);</code>
Function descriptions	set the read command
Precondition	-
The called functions	-
Input parameter{in}	
<code>read_command_mode</code>	configure SPI PSRAM read command mode
<code>EXMC_SQPIPSRAM_R_EAD_MODE_DISABLE</code>	not SPI mode
<code>EXMC_SQPIPSRAM_R_EAD_MODE_SPI</code>	SPI mode
<code>EXMC_SQPIPSRAM_R_EAD_MODE_SQPI</code>	SQPI mode
<code>EXMC_SQPIPSRAM_R_EAD_MODE_QPI</code>	QPI mode
Input parameter{in}	
<code>read_wait_cycle</code>	wait cycle number after address phase,0..15
Input parameter{in}	
<code>read_command_code</code>	read command for AHB read transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* set the SQPIPSRAM read command */

exmc_sqipsram_read_command_set(EXMC_SQPIPSRAM_READ_MODE_SQPI,1,1);

```

exmc_sqipsram_write_command_set

The description of exmc_sqipsram_write_command_set is shown as below:

Table 3-356. Function exmc_sqipsram_write_command_set

Function name	exmc_sqipsram_write_command_set
Function prototype	void exmc_sqipsram_write_command_set(uint32_t write_command_mode,uint32_t write_wait_cycle,uint32_t write_command_code);
Function descriptions	set the write command
Precondition	-
The called functions	-
Input parameter{in}	
write_command_mod e	configure SPI PSRAM write command mode
<i>EXMC_SQPIPSRAM_WRITE_MODE_DISABLE</i>	not SPI mode
<i>EXMC_SQPIPSRAM_WRITE_MODE_SPI</i>	SPI mode
<i>EXMC_SQPIPSRAM_WRITE_MODE_SQPI</i>	SQPI mode
<i>EXMC_SQPIPSRAM_WRITE_MODE_QPI</i>	QPI mode
Input parameter{in}	
write_wait_cycle	wait cycle number after address phase,0..15
Input parameter{in}	
write_command_code	write command for AHB write transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the SQPIPSRAM write command */
exmc_sqipsram_write_command_set(EXMC_SQPIPSRAM_WRITE_MODE_SQPI,1,1);
```

exmc_sqipsram_read_id_command_send

The description of exmc_sqipsram_read_id_command_send is shown as below:

Table 3-357. Function exmc_sqipsram_read_id_command_send

Function name	exmc_sqipsram_read_id_command_send
Function prototype	void exmc_sqipsram_read_id_command_send(void);

Function descriptions	send SPI read ID command
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* send SPI read ID command */

exmc_sqipssram_read_id_command_send();
```

exmc_sqipssram_write_cmd_send

The description of exmc_sqipssram_write_cmd_send is shown as below:

Table 3-358. Function exmc_sqipssram_write_cmd_send

Function name	exmc_sqipssram_write_cmd_send
Function prototype	void exmc_sqipssram_write_cmd_send(void);
Function descriptions	send SPI special command which does not have address and data phase
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* send SPI special command which does not have address and data phase */

exmc_sqipssram_write_cmd_send();
```

exmc_sqipssram_low_id_get

The description of exmc_sqipssram_low_id_get is shown as below:

Table 3-359. Function exmc_sqipssram_low_id_get

Function name	exmc_sqipssram_low_id_get
Function prototype	uint32_t exmc_sqipssram_low_id_get(void);
Function descriptions	get the EXMC SPI ID low data

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the ID low data

Example:

```

/* the ID low data */

uint32_t id;

id = exmc_sqpipsram_low_id_get();

```

exmc_sqpipsram_high_id_get

The description of exmc_sqpipsram_high_id_get is shown as below:

Table 3-360. Function exmc_sqpipsram_low_id_get

Function name	exmc_sqpipsram_high_id_get
Function prototype	uint32_t exmc_sqpipsram_high_id_get(void);
Function descriptions	get the EXMC SPI ID high data
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the ID high data

Example:

```

/* the ID high data */

uint32_t id;

id = exmc_sqpipsram_high_id_get();

```

exmc_sqpipsram_send_command_state_get

The description of exmc_sqpipsram_send_command_state_get is shown as below:

Table 3-361. Function exmc_sqpipsram_send_command_state_get

Function name	exmc_sqpipsram_send_command_state_get
----------------------	---------------------------------------

Function prototype	FlagStatus exmc_sqipsram_send_command_state_get(uint32_t send_command_flag);
Function descriptions	get the bit value of EXMC send write command bit or read ID command
Precondition	-
The called functions	-
Input parameter{in}	
send_command_flag	the send command flag
<i>EXMC_SEND_COMMAND_FLAG_RID</i>	EXMC_SRCMD_RDID flag bit
<i>EXMC_SEND_COMMAND_FLAG_SC</i>	EXMC_SWCMD_SC flag bit
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```

/* check EXMC_SEND_COMMAND_FLAG_SC is set or not*/
if(RESET !=exmc_sqipsram_send_command_state_get(EXMC_SEND_COMMAND_FLAG_SC));

```

exmc_interrupt_enable

The description of `exmc_interrupt_enable` is shown as below:

Table 3-362. Function `exmc_interrupt_enable`

Function name	exmc_interrupt_enable
Function prototype	void exmc_interrupt_enable(uint32_t exmc_bank,uint32_t interrupt);
Function descriptions	enable EXMC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
exmc_bank	specifies the NAND bank , PC card bank or SDRAM device
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA_RD</i>	the PC Card bank
<i>EXMC_SDRAM_DEVICE_E0</i>	the SDRAM device0
<i>EXMC_SDRAM_DEVICE_E1</i>	the SDRAM device1
Input parameter{in}	
interrupt	specify get which interrupt flag

<i>EXMC_NAND_PCCAR_D_INT_FLAG_RISE</i>	rising edge interrupt and flag
<i>EXMC_NAND_PCCAR_D_INT_FLAG_LEVEL</i>	high-level interrupt and flag
<i>EXMC_NAND_PCCAR_D_INT_FLAG_FALL</i>	falling edge interrupt and flag
<i>EXMC_SDRAM_INT_FLAG_REFRESH</i>	refresh error interrupt and flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable EXMC interrupt*/
exmc_interrupt_enable(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

exmc_interrupt_disable

The description of exmc_interrupt_disable is shown as below:

Table 3-363. Function exmc_interrupt_disable

Function name	exmc_interrupt_disable
Function prototype	void exmc_interrupt_disable(uint32_t exmc_bank,uint32_t interrupt);
Function descriptions	disable EXMC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
<i>exmc_bank</i>	specifies the NAND bank , PC card bank or SDRAM device
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA_RD</i>	the PC Card bank
<i>EXMC_SDRAM_DEVIC_E0</i>	the SDRAM device0
<i>EXMC_SDRAM_DEVIC_E1</i>	the SDRAM device1
Input parameter{in}	
interrupt	specify get which interrupt flag
<i>EXMC_NAND_PCCAR_D_INT_FLAG_RISE</i>	rising edge interrupt and flag
<i>EXMC_NAND_PCCAR_D_INT_FLAG_LEVEL</i>	high-level interrupt and flag

<i>EXMC_NAND_PCCAR_D_INT_FLAG_FALL</i>	falling edge interrupt and flag
<i>EXMC_SDRAM_INT_FLAG_REFRESH</i>	refresh error interrupt and flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable EXMC interrupt*/
exmc_interrupt_disable(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

exmc_flag_get

The description of exmc_flag_get is shown as below:

Table 3-364. Function exmc_flag_get

Function name	exmc_flag_get
Function prototype	FlagStatus exmc_flag_get(uint32_t exmc_bank,uint32_t flag);
Function descriptions	get EXMC flag status
Precondition	-
The called functions	-
Input parameter{in}	
exmc_bank	specifies the NAND bank , PC card bank or SDRAM device
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA_RD</i>	the PC Card bank
<i>EXMC_SDRAM_DEVICE0</i>	the SDRAM device0
<i>EXMC_SDRAM_DEVICE1</i>	the SDRAM device1
Input parameter{in}	
flag	specify get which flag
<i>EXMC_NAND_PCCAR_D_FLAG_RISE</i>	interrupt rising edge status
<i>EXMC_NAND_PCCAR_D_FLAG_LEVEL</i>	interrupt high-level status
<i>EXMC_NAND_PCCAR_D_FLAG_FALL</i>	interrupt falling edge status
<i>EXMC_SDRAM_FLAG</i>	refresh error interrupt flag

<i>_REFRESH</i>	
<i>EXMC_SDRAM_FLAG_NREADY</i>	not ready status
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check EXMC_NAND_PCCARD_FLAG_RISE is set or not*/
if(RESET != exmc_flag_get(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_FLAG_RISE));
```

exmc_flag_clear

The description of `exmc_flag_clear` is shown as below:

Table 3-365. Function `exmc_flag_clear`

Function name	exmc_flag_clear
Function prototype	FlagStatus exmc_flag_clear (uint32_t exmc_bank,uint32_t flag);
Function descriptions	clear EXMC flag status
Precondition	-
The called functions	-
Input parameter{in}	
exmc_bank	specifies the NAND bank , PC card bank or SDRAM device
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA_RD</i>	the PC Card bank
<i>EXMC_SDRAM_DEVIC_E0</i>	the SDRAM device0
<i>EXMC_SDRAM_DEVIC_E1</i>	the SDRAM device1
Input parameter{in}	
flag	specify get which flag
<i>EXMC_NAND_PCCAR_D_FLAG_RISE</i>	interrupt rising edge status
<i>EXMC_NAND_PCCAR_D_FLAG_LEVEL</i>	interrupt high-level status
<i>EXMC_NAND_PCCAR_D_FLAG_FALL</i>	interrupt falling edge status
<i>EXMC_SDRAM_FLAG_REFRESH</i>	refresh error interrupt flag

<i>EXMC_SDRAM_FLAG_NREADY</i>	not ready status
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EXMC flag status */

exmc_flag_clear(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_FLAG_RISE);
```

exmc_interrupt_flag_get

The description of `exmc_interrupt_flag_get` is shown as below:

Table 3-366. Function exmc_interrupt_flag_get

Function name	exmc_interrupt_flag_get
Function prototype	FlagStatus exmc_interrupt_flag_get(uint32_t exmc_bank,uint32_t interrupt);
Function descriptions	get EXMC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
<i>exmc_bank</i>	specifies the NAND bank , PC card bank or SDRAM device
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA_RD</i>	the PC Card bank
<i>EXMC_SDRAM_DEVICE_E0</i>	the SDRAM device0
<i>EXMC_SDRAM_DEVICE_E1</i>	the SDRAM device1
Input parameter{in}	
interrupt	specify get which interrupt flag
<i>EXMC_NAND_PCCARD_INT_FLAG_RISE</i>	rising edge interrupt and flag
<i>EXMC_NAND_PCCARD_INT_FLAG_LEVEL</i>	high-level interrupt and flag
<i>EXMC_NAND_PCCARD_INT_FLAG_FALL</i>	falling edge interrupt and flag
<i>EXMC_SDRAM_INT_FLAG_REFRESH</i>	refresh error interrupt and flag
Output parameter{out}	
-	-

Return value	
FlagStatus	SET or RESET

Example:

```
/* check EXMC_NAND_PCCARD_INT_FLAG_RISE is set or not*/
if(RESET != exmc_interrupt_flag_get(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_FLAG_RISE));
```

exmc_interrupt_flag_clear

The description of `exmc_interrupt_flag_clear` is shown as below:

Table 3-367. Function `exmc_interrupt_flag_clear`

Function name	exmc_interrupt_flag_clear
Function prototype	void exmc_interrupt_flag_clear(uint32_t exmc_bank,uint32_t interrupt);
Function descriptions	clear EXMC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
<code>exmc_bank</code>	specifies the NAND bank , PC card bank or SDRAM device
<code>EXMC_BANK1_NAND</code>	the NAND bank1
<code>EXMC_BANK2_NAND</code>	the NAND bank2
<code>EXMC_BANK3_PCCA_RD</code>	the PC Card bank
<code>EXMC_SDRAM_DEVIC_E0</code>	the SDRAM device0
<code>EXMC_SDRAM_DEVIC_E1</code>	the SDRAM device1
Input parameter{in}	
<code>interrupt</code>	specify get which interrupt flag
<code>EXMC_NAND_PCCARD_INT_FLAG_RISE</code>	rising edge interrupt and flag
<code>EXMC_NAND_PCCARD_INT_FLAG_LEVEL</code>	high-level interrupt and flag
<code>EXMC_NAND_PCCARD_INT_FLAG_FALL</code>	falling edge interrupt and flag
<code>EXMC_SDRAM_INTFLAG_REFRESH</code>	refresh error interrupt and flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EXMC interrupt flag */

exmc_interrupt_flag_clear(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

3.12. EXTI

EXTI is the interrupt/event controller in the MCU. It contains up to 23 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.12.1](#), the EXTI firmware functions are introduced in chapter [3.12.2](#)

3.12.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

Table 3-368. EXTI Registers

Registers	Descriptions
EXTI_INTEN	interrupt enable register
EXTI_EVENT	event enable register
EXTI_RTEN	rising edge trigger enable register
EXTI_FTEN	falling edge trigger enable register
EXTI_SWIEV	software interrupt event register
EXTI_PD	pending register

3.12.2. Descriptions of Peripheral functions

EXTI firmware functions are listed in the table shown as below:

Table 3-369. EXTI firmware function

Function name	Function description
exti_deinit	deinitialize the EXTI
exti_init	initialize EXTI line x
exti_interrupt_enable	enable the interrupts from EXTI line x
exti_interrupt_disable	disable the interrupts from EXTI line x
exti_event_enable	enable the events from EXTI line x
exti_event_disable	disable the events from EXTI line x
exti_software_interrupt_enable	enable the software interrupt event from EXTI line x
exti_software_interrupt_disable	disable the software interrupt event from EXTI line x
exti_flag_get	get EXTI line x interrupt pending flag
exti_flag_clear	clear EXTI line x interrupt pending flag
exti_interrupt_flag_get	get EXTI line x interrupt pending flag
exti_interrupt_flag_clear	clear EXTI line x interrupt pending flag

Enum exti_line_enum

Table 3-370. Enum exti_line_enum

Member name	Function description
EXTI_0	EXTI line 0
EXTI_1	EXTI line 1
EXTI_2	EXTI line 2
EXTI_3	EXTI line 3
EXTI_4	EXTI line 4
EXTI_5	EXTI line 5
EXTI_6	EXTI line 6
EXTI_7	EXTI line 7
EXTI_8	EXTI line 8
EXTI_9	EXTI line 9
EXTI_10	EXTI line 10
EXTI_11	EXTI line 11
EXTI_12	EXTI line 12
EXTI_13	EXTI line 13
EXTI_14	EXTI line 14
EXTI_15	EXTI line 15
EXTI_16	EXTI line 16
EXTI_17	EXTI line 17
EXTI_18	EXTI line 18
EXTI_19	EXTI line 19
EXTI_20	EXTI line 20
EXTI_21	EXTI line 21
EXTI_22	EXTI line 22

Enum exti_mode_enum

Table 3-371. Enum exti_mode_enum

Member name	Function description
EXTI_INTERRUPT	EXTI interrupt mode
EXTI_EVENT	EXTI event mode

Enum exti_trig_type_enum

Table 3-372. Enum exti_trig_type_enum

Member name	Function description
EXTI_TRIG_RISING	EXTI rising edge trigger
EXTI_TRIG_FALLING	EXTI falling edge trigger
EXTI_TRIG_BOTH	EXTI rising and falling edge trigger
EXTI_TRIG_NONE	EXTI without rising or falling edge trigger

exti_deinit

The description of exti_deinit is shown as below:

Table 3-373. Function exti_deinit

Function name	exti_deinit
Function prototype	void exti_deinit(void);
Function descriptions	deinitialize the EXTI
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize the EXTI */

exti_deinit();
```

exti_init

The description of exti_init is shown as below:

Table 3-374. Function exti_init

Function name	exti_init
Function prototype	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
Function descriptions	initialize EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-370. Enum exti_line_enum
Input parameter{in}	
mode	EXTI mode, refer to Table 3-371. Enum exti_mode_enum
Input parameter{in}	
trig_type	trigger type, refer to Table 3-372. Enum exti_trig_type_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure EXTI_0 */

exti_init(EXTI_0, EXTI_INTERRUPT, EXTI_TRIG_BOTH);
```

exti_interrupt_enable

The description of exti_interrupt_enable is shown as below:

Table 3-375. Function exti_interrupt_enable

Function name	exti_interrupt_enable
Function prototype	void exti_interrupt_enable(exti_line_enum linex);
Function descriptions	enable the interrupts from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-370. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the interrupts from EXTI line 0 */

exti_interrupt_enable(EXTI_0);
```

exti_interrupt_disable

The description of exti_interrupt_disable is shown as below:

Table 3-376. Function exti_interrupt_disable

Function name	exti_interrupt_disable
Function prototype	void exti_interrupt_disable(exti_line_enum linex);
Function descriptions	disable the interrupts from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-370. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the interrupts from EXTI line 0 */
```

```
exti_interrupt_disable(EXTI_0);
```

exti_event_enable

The description of exti_event_enable is shown as below:

Table 3-377. Function exti_event_enable

Function name	exti_event_enable
Function prototype	void exti_event_enable(exti_line_enum linex);
Function descriptions	enable the events from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-370. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the events from EXTI line 0 */
```

```
exti_event_enable(EXTI_0);
```

exti_event_disable

The description of exti_event_disable is shown as below:

Table 3-378. Function exti_event_disable

Function name	exti_event_disable
Function prototype	void exti_event_disable(exti_line_enum linex);
Function descriptions	disable the events from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-370. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the events from EXTI line 0 */
```

```
exti_event_disable(EXTI_0);
```

exti_software_interrupt_enable

The description of exti_software_interrupt_enable is shown as below:

Table 3-379. Function exti_software_interrupt_enable

Function name	exti_software_interrupt_enable
Function prototype	void exti_software_interrupt_enable(exti_line_enum linex);
Function descriptions	enable the software interrupt event from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-370. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable EXTI line 0 software interrupt */

exti_software_interrupt_enable(EXTI_0);
```

exti_software_interrupt_disable

The description of exti_software_interrupt_disable is shown as below:

Table 3-380. Function exti_software_interrupt_disable

Function name	exti_software_interrupt_disable
Function prototype	void exti_software_interrupt_disable(exti_line_enum linex);
Function descriptions	disable the software interrupt event EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-370. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable EXTI line 0 software interrupt */

exti_software_interrupt_disable(EXTI_0);
```

exti_flag_get

The description of exti_flag_get is shown as below:

Table 3-381. Function exti_flag_get

Function name	exti_flag_get
Function prototype	FlagStatus exti_flag_get(exti_line_enum linex);
Function descriptions	get EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-370. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get EXTI line 0 flag status */
FlagStatus state = exti_flag_get(EXTI_0);
```

exti_flag_clear

The description of exti_flag_clear is shown as below:

Table 3-382. Function exti_flag_clear

Function name	exti_flag_clear
Function prototype	void exti_flag_clear(exti_line_enum linex);
Function descriptions	clear EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-370. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EXTI line 0 flag status */
exti_flag_clear(EXTI_0);
```

exti_interrupt_flag_get

The description of exti_interrupt_flag_get is shown as below:

Table 3-383. Function exti_interrupt_flag_get

Function name	exti_interrupt_flag_get
Function prototype	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
Function descriptions	get EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-370. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get EXTI line 0 interrupt flag status */
FlagStatus state = exti_interrupt_flag_get(EXTI_0);
```

exti_interrupt_flag_clear

The description of exti_interrupt_flag_clear is shown as below:

Table 3-384. Function exti_interrupt_flag_clear

Function name	exti_interrupt_flag_clear
Function prototype	void exti_interrupt_flag_clear(exti_line_enum linex);
Function descriptions	clear EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-370. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EXTI line 0 interrupt flag status */
exti_interrupt_flag_clear(EXTI_0);
```

3.13. FMC

There is flash controller and option byte for GD32F4xx series. The FMC registers are listed in chapter [3.13.1](#), the FMC firmware functions are introduced in chapter [3.13.2](#).

3.13.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

Table 3-385. FMC Registers

Registers	Descriptions
FMC_WS	Wait state register
FMC_KEY	Unlock key register
FMC_OBKEY	Option byte unlock key register
FMC_STAT	Status register
FMC_CTL	Control register
FMC_OBCTL0	Option byte control register 0
FMC_OBCTL1	Option byte control register 1
FMC_PECFG	Page erase configuration register
FMC_PEKEY	Unlock page erase key register
FMC_WSEN	Wait state enable register
FMC_PID	Product ID register

3.13.2. Descriptions of Peripheral functions

FMC firmware functions are listed in the table shown as below:

Table 3-386. FMC firmware function

Function name	Function description
fmc_wscnt_set	set the FMC wait state counter
fmc_unlock	unlock the main FMC operation
fmc_lock	lock the main FMC operation
fmc_page_erase	FMC erase page
fmc_sector_erase	FMC erase sector
fmc_mass_erase	FMC erase whole chip
fmc_bank0_erase	FMC erase whole bank0
fmc_bank1_erase	FMC erase whole bank1
fmc_word_program	FMC program a word at the corresponding address
fmc_halfword_program	FMC program a half word at the corresponding address
fmc_byte_program	FMC program a byte at the corresponding address
ob_unlock	unlock the option byte operation
ob_lock	lock the option byte operation
ob_start	send option byte change command

Function name	Function description
ob_erase	erase and reset option byte
ob_write_protection_enable	enable write protect
ob_write_protection_disable	disable write protect
ob_drp_enable	enable erase/program protection and D-bus read protection
ob_drp_disable	disable erase/program protection and D-bus read protection
ob_security_protection_config	set the option byte security protection level
ob_user_write	write the FMC option byte user
ob_user_bor_threshold	option byte BOR threshold value
ob_boot_mode_config	configure the boot mode
ob_user_get	get the FMC option byte user
ob_write_protection0_get	get the FMC option byte write protection
ob_write_protection1_get	get the FMC option byte write protection
ob_drp0_get	get the FMC erase/program protection and D-bus read protection option bytes value
ob_drp1_get	get the FMC erase/program protection and D-bus read protection option bytes value
ob_spc_get	get option byte security protection code value
ob_user_bor_threshold_get	get the FMC threshold value
fmc_flag_get	get flag set or reset
fmc_flag_clear	clear the FMC pending flag
fmc_interrupt_enable	enable FMC interrupt
fmc_interrupt_disable	disable FMC interrupt
fmc_interrupt_flag_get	get interrupt flag set or reset
fmc_interrupt_flag_clear	clear the FMC pending interrupt flag
fmc_state_get	return the FMC state
fmc_ready_wait	check FMC ready or not

fmc_state_enum

Table 3-387. fmc_state_enum

enum name	enum description
FMC_READY	the operation has been completed
FMC_BUSY	the operation is in progress
FMC_RDDERR	read D-bus protection error
FMC_PGSERR	program sequence error
FMC_PGMERR	program size not match error
FMC_WPERR	erase/program protection error
FMC_OPERR	operation error
FMC_TOERR	timeout error

fmc_wscnt_set

The description of fmc_wscnt_set is shown as below:

Table 3-388. Function fmc_wscnt_set

Function name	fmc_wscnt_set
Function prototype	void fmc_wscnt_set(uint32_t wscnt);
Function descriptions	set the wait state counter value
Precondition	-
The called functions	-
Input parameter{in}	
wscnt	wait state counter value
WS_WSCNT_x	FMC X(X = 0~15) wait
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the wait state counter value */
fmc_wscnt_set (WS_WSCNT_1);
```

fmc_unlock

The description of fmc_unlock is shown as below:

Table 3-389. Function fmc_unlock

Function name	fmc_unlock
Function prototype	void fmc_unlock (void);
Function descriptions	unlock the main FMC operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock the main FMC operation */
fmc_unlock ( );
```

fmc_lock

The description of fmc_lock is shown as below:

Table 3-390. Function fmc_lock

Function name	fmc_lock
Function prototype	void fmc_lock(void);
Function descriptions	lock the main FMC operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock the main FMC operation */

fmc_lock();
```

fmc_page_erase

The description of fmc_page_erase is shown as below:

Table 3-391. Function fmc_page_erase

Function name	fmc_page_erase
Function prototype	fmc_state_enum fmc_page_erase(uint32_t page_addr);
Function descriptions	erase page
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
page_addr	page address
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to <u>fmc_state_enum</u>

Example:

```
/* erase page */

fmc_unlock();

fmc_state_enum state = FMC_READY;
```

```
state = fmc_page_erase(0x08008000);
```

fmc_sector_erase

The description of fmc_sector_erase is shown as below:

Table 3-392. Function fmc_sector_erase

Function name	fmc_sector_erase
Function prototype	fmc_state_enum fmc_sector_erase(uint32_t fmc_sector);
Function descriptions	erase sector
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
fmc_sector	select the sector to erase
CTL_SECTOR_NUMB ER_x	sector X(X = 0~27)
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to fmc state enum

Example:

```
/* erase number 15 sector */

fmc_unlock();

fmc_state_enum state = FMC_READY;

state = fmc_sector_erase(CTL_SECTOR_NUMBER_15);
```

fmc_mass_erase

The description of fmc_mass_erase is shown as below:

Table 3-393. Function fmc_mass_erase

Function name	fmc_mass_erase
Function prototype	fmc_state_enum fmc_mass_erase(void);
Function descriptions	erase whole chip
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to fmc state enum

Example:

```
/* erase the whole chip */

fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_mass_erase();
```

fmc_bank0_erase

The description of fmc_bank0_erase is shown as below:

Table 3-394. Function fmc_bank0_erase

Function name	fmc_bank0_erase
Function prototype	fmc_state_enum fmc_bank0_erase(void)
Function descriptions	erase all FMC sectors in bank0
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to fmc state enum

Example:

```
/* erase bank0 */

fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_bank0_erase( );
```

fmc_bank1_erase

The description of fmc_bank1_erase is shown as below:

Table 3-395. Function fmc_bank1_erase

Function name	fmc_bank1_erase
Function prototype	fmc_state_enum fmc_bank1_erase(void)
Function descriptions	erase all FMC sectors in bank0
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to <u>fmc state enum</u>

Example:

```

/* erase bank1 */

fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_bank1_erase( );

```

fmc_word_program

The description of fmc_word_program is shown as below:

Table 3-396. Function fmc_word_program

Function name	fmc_word_program
Function prototype	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
Function descriptions	program a word at the corresponding address
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
address	the address to program
data	word to program(0x00000000 - 0xFFFFFFFF)
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to <u>fmc state enum</u>

Example:

```

fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_word_program(0x08004000, 0x12345678);

```

fmc_halfword_program

The description of fmc_halfword_program is shown as below:

Table 3-397. Function fmc_halfword_program

Function name	fmc_halfword_program
Function prototype	fmc_state_enum fmc_halfword_program(uint32_t address, uint16_t data);
Function descriptions	program a halfword at the corresponding address

Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
address	the address to program
data	halfword to program(0x0000 - 0xFFFF)
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to <u>fmc_state_enum</u>

Example:

```

/* program half word at the corresponding address */

fmc_unlock();

fmc_state_enum state = FMC_READY;

state = fmc_halfword_program (0x08004000, 0x1234);

```

fmc_byte_program

The description of fmc_byte_program is shown as below:

Table 3-398. Function fmc_halfword_program

Function name	fmc_halfword_program
Function prototype	fmc_state_enum fmc_byte_program(uint32_t address, uint8_t data);
Function descriptions	program a byte at the corresponding address
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
address	the address to program
data	byte to program(0x00 - 0xFF)
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to <u>fmc_state_enum</u>

Example:

```

/* program a byte at the corresponding address */

fmc_unlock();

fmc_state_enum state = FMC_READY;

state = fmc_byte_program (0x08004000, 0x12);

```

ob_unlock

The description of ob_unlock is shown as below:

Table 3-399. Function ob_unlock

Function name	ob_unlock
Function prototype	void ob_unlock(void);
Function descriptions	unlock the option byte operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock the option byte operation */

ob_unlock( );
```

ob_lock

The description of ob_lock is shown as below:

Table 3-400. Function ob_lock

Function name	ob_lock
Function prototype	void ob_lock(void);
Function descriptions	lock the option byte operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock the option byte operation */

ob_lock( );
```

ob_start

The description of ob_start is shown as below:

Table 3-401. Function ob_start

Function name	ob_start
Function prototype	void ob_start(void);
Function descriptions	send option byte change command
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable write protection and then send the start command */
ob_unlock( );
ob_write_protection_enable (OB_WP7);
ob_start();
```

ob_erase

The description of ob_erase is shown as below:

Table 3-402. Function ob_erase

Function name	ob_erase
Function prototype	void ob_erase(void);
Function descriptions	erase and reset option byte
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* erase and reset option byte */
```

```

ob_unlock( );
ob_erase( );
ob_start( );
ob_lock( );

```

ob_write_protection_enable

The description of ob_write_protection_enable is shown as below:

Table 3-403. Function ob_write_protection_enable

Function name	ob_write_protection_enable
Function prototype	ErrStatus ob_write_protection_enable(uint32_t ob_wp);
Function descriptions	enable write protection
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
ob_wp	specify sector to be write protected
<i>OB_WP_x</i>	sector x(x = 0,1,2...22)
<i>OB_WP_23_27</i>	sector23~27
<i>OB_WP_ALL</i>	all sector
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```

/* enable write protection of sector 7 */

ErrStatus temp = ERROR;

ob_unlock( );

temp = ob_write_protection_enable (OB_WP7);

ob_start();

ob_lock( );

```

ob_write_protection_disable

The description of ob_write_protection_disable is shown as below:

Table 3-404. Function ob_write_protection_disable

Function name	ob_write_protection_disable
Function prototype	ErrStatus ob_write_protection_disable(uint32_t ob_wp);
Function descriptions	disable write protection

Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
ob_wp	specify sector to be write protected
<i>OB_WP_x</i>	sector x(x = 0,1,2...22)
<i>OB_WP_23_27</i>	sector23~27
<i>OB_WP_ALL</i>	all sector
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* disable write protection of sector 7 */

ErrStatus temp = ERROR;

ob_unlock( );

temp = ob_write_protection_disable (OB_WP7);

ob_start();

ob_lock( );
```

ob_drp_enable

The description of **ob_drp_enable** is shown as below:

Table 3-405. Function ob_drp_enable

Function name	ob_drp_enable
Function prototype	void ob_drp_enable(uint32_t ob_drp);
Function descriptions	enable erase/program protection and D-bus read protection
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
ob_drp	enable the WPx bits used as erase/program protection and D-bus read protection of each sector
<i>OB_DRP_x</i>	sector x(x = 0,1,2...22)
<i>OB_DRP_23_27</i>	sector23~27
<i>OB_DRP_ALL</i>	all sector
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable write protection and D-BUS read protectio of sector 7 */
```

```
ob_unlock( );
ob_drp_enable (OB_DRP7);
ob_start();
ob_lock( );
```

ob_drp_disable

The description of ob_drp_disable is shown as below:

Table 3-406. Function ob_drp_disable

Function name	ob_drp_disable
Function prototype	void ob_drp_disable(uint32_t ob_drp);
Function descriptions	disable erase/program protection and D-bus read protection
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
ob_drp	disable the WPx bits used as erase/program protection and D-bus read protection of each sector
OB_DRP_x	sector x(x = 0,1,2...22)
OB_DRP_23_27	sector23~27
OB_DRP_ALL	all sector
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable write protection and D-BUS read protectio of sector 7 */
```

```
ob_unlock( );
ob_drp_disable (OB_DRP7);
ob_start();
ob_lock( );
```

ob_security_protection_config

The description of ob_security_protection_config is shown as below:

Table 3-407. Function ob_security_protection_config

Function name	ob_security_protection_config
Function prototype	void ob_security_protection_config(uint8_t ob_spc);

Function descriptions	configure security protection level
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
ob_spc	specify security protection
<i>FMC_NSPC</i>	no security protection
<i>FMC_LSPC</i>	low security protection
<i>FMC_HSPC</i>	high security protection
Output parameter{out}	
-	-
Return value	
-	state of FMC, refer to <i>fmc_state_enum</i>

Example:

```

/* config FMC as low security protection */

ob_unlock( );

ob_security_protection_config(FMC_LSPC);

ob_start( );

ob_lock( );

```

ob_user_write

The description of ob_user_write is shown as below:

Table 3-408. Function ob_user_write

Function name	ob_user_write
Function prototype	void ob_user_write(uint32_t ob_fwdgt, uint32_t ob_deepsleep, uint32_t ob_stdby);
Function descriptions	program the FMC user option byte
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
ob_fwdgt	option byte watchdog value
<i>OB_FWDGT_SW</i>	software free watchdog
<i>OB_FWDGT_HW</i>	hardware free watchdog
Input parameter{in}	
ob_deepsleep	option byte deepsleep reset value
<i>OB_DEEPSLEEP_NRS_T</i>	no reset when entering deepsleep mode
<i>OB_DEEPSLEEP_RST</i>	generate a reset instead of entering deepsleep mode
Input parameter{in}	

ob_stby	option byte standby reset value
<i>OB_STDBY_NRST</i>	no reset when entering standby mode
<i>OB_STDBY_RST</i>	generate a reset instead of entering standby mode
Output parameter{out}	
-	-
Return value	
-	state of FMC, refer to <i>fmc state enum</i>

Example:

```

/* config the USER byte of option byte */

ob_unlock( );

ob_user_write(OB_FWDGT_SW, OB_DEEPSLEEP_NRST, OB_STDBY_NRST);

ob_start ( );

ob_lock( );

```

ob_user_bor_threshold

The description of ob_user_bor_threshold is shown as below:

Table 3-409. Function ob_user_bor_threshold

Function name	ob_user_bor_threshold
Function prototype	void ob_user_bor_threshold(uint32_t ob_bor_th);
Function descriptions	program the option byte BOR threshold value
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
ob_bor_th	user option byte
<i>OB_BOR_TH_VALUE3</i>	BOR threshold value 3
<i>OB_BOR_TH_VALUE2</i>	BOR threshold value 2
<i>OB_BOR_TH_VALUE1</i>	BOR threshold value 1
<i>OB_BOR_TH_OFF</i>	no BOR function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* config the BOR threshold value as OB_BOR_TH_OFF */

ob_unlock( );

ob_user_bor_threshold(OB_BOR_TH_OFF);

```

```
ob_start( );
ob_lock( );
```

ob_boot_mode_config

The description of ob_boot_mode_config is shown as below:

Table 3-410. Function ob_boot_mode_config

Function name	ob_boot_mode_config
Function prototype	void ob_boot_mode_config(uint32_t boot_mode);
Function descriptions	configure the option byte boot bank value
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
boot_mode	specifies the option byte boot bank value
<i>OB_BB_DISABLE</i>	boot from bank0
<i>OB_BB_ENABLE</i>	boot from bank1 or bank0 if bank1 is void
Output parameter{out}	
-	-
Return value	
-	state of FMC, refer to <i>fmc_state_enum</i>

Example:

```
/* config boot from bank0 */
ob_unlock( );

ob_boot_mode_config(OB_BB_DISABLE);

ob_start( );

ob_lock( );
```

ob_user_get

The description of ob_user_get is shown as below:

Table 3-411. Function ob_user_get

Function name	ob_user_get
Function prototype	uint8_t ob_user_get(void);
Function descriptions	get the FMC user option byte
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

Return value	
uint8_t	the FMC user option byte values: ob_fwdgt(Bit0), ob_deepsleep(Bit1), ob_stdby(Bit2)

Example:

```
/* get the the FMC user option byte values */
uint8_t ob_user_values = 0;
ob_user_values = ob_user_get( );
```

ob_write_protection0_get

The description of **ob_write_protection0_get** is shown as below:

Table 3-412. Function ob_write_protection0_get

Function name	ob_write_protection0_get
Function prototype	uint16_t ob_write_protection0_get(void);
Function descriptions	get the FMC option byte write protection of bank0
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
Uint16_t	the FMC write protection option byte value

Example:

```
/* get the FMC write protection option byte value */
uint16_t bank0_protection_value = 0;
bank0_protection_value = ob_write_protection0_get( );
```

ob_write_protection1_get

The description of **ob_write_protection1_get** is shown as below:

Table 3-413. Function ob_write_protection1_get

Function name	ob_write_protection1_get
Function prototype	uint16_t ob_write_protection1_get(void);
Function descriptions	get the FMC option byte write protection of bank1
Precondition	-
The called functions	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
Uint16_t	the FMC write protection option byte value

Example:

```
/* get the FMC write protection option byte value */

uint16_t bank1_protection_value = 0;

bank1_protection_value = ob_write_protection1_get( );
```

ob_drp0_get

The description of **ob_drp0_get** is shown as below:

Table 3-414. Function ob_drp0_get

Function name	ob_drp0_get
Function prototype	uint16_t ob_drp0_get(void);
Function descriptions	get the FMC D-bus read protection protection of bank0
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	the FMC erase/program protection and D-bus read protection option bytes value

Example:

```
/* get the FMC erase/program protection and D-bus read protection option bytes value */

uint16_t bank0_drp_value = 0;

bank0_drp_value = ob_drp0_get( );
```

ob_drp1_get

The description of **ob_drp1_get** is shown as below:

Table 3-415. Function ob_drp1_get

Function name	ob_drp1_get
Function prototype	uint16_t ob_drp1_get(void);

Function descriptions	get the FMC D/bus read protection protection of bank1
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	the FMC erase/program protection and D/bus read protection option bytes value

Example:

```
/* get the FMC erase/program protection and D/bus read protection option bytes value */
uint16_t bank1_drp_value = 0;
bank1_drp_value = ob_drp1_get( );
```

ob_spc_get

The description of ob_spc_get is shown as below:

Table 3-416. Function ob_spc_get

Function name	ob_spc_get
Function prototype	FlagStatus ob_spc_get(void);
Function descriptions	get the FMC option byte security protection
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the FMC option byte security protection */
FlagStatus spc_value = RESET;
spc_value = ob_spc_get();
```

ob_user_bor_threshold_get

The description of ob_user_bor_threshold_get is shown as below:

Table 3-417. Function ob_user_bor_threshold_get

Function name	ob_user_bor_threshold_get
Function prototype	uint8_t ob_user_bor_threshold_get(void);
Function descriptions	get the FMC option byte BOR threshold value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint8_t	the FMC BOR threshold value:OB_BOR_TH_OFF,OB_BOR_TH_VALUE1,OB_BOR_TH_VALUE2, OB_BOR_TH_VALUE3

Example:

```

/* get the FMC option byte BOR threshold value */

uint8_t bor_value = 0;

bor_value = ob_user_bor_threshold_get( );

```

fmc_flag_get

The description of fmc_flag_get is shown as below:

Table 3-418. Function fmc_flag_get

Function name	fmc_flag_get
Function prototype	FlagStatus fmc_flag_get(uint32_t flag);
Function descriptions	check flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
flag	check FMC flag
<i>FMC_FLAG_BUSY</i>	FMC busy flag bit
<i>FMC_FLAG_RDDERR</i>	FMC read D-bus protection error flag bit
<i>FMC_FLAG_PGSERR</i>	FMC program sequence error flag bit
<i>FMC_FLAG_PGMERR</i>	FMC program size not match error flag bit
<i>FMC_FLAG_WPERR</i>	FMC Erase/Program protection error flag bit
<i>FMC_FLAG_OPERR</i>	FMC operation error flag bit
<i>FMC_FLAG_END</i>	FMC end of operation flag bit
Output parameter{out}	
-	-
Return value	

FlagStatus	SET or RESET
------------	--------------

Example:

```
/* check the FMC_FLAG_END flag set or not*/
```

```
FlagStatus flag = RESET;
```

```
flag = fmc_flag_get(FMC_FLAG_END);
```

fmc_flag_clear

The description of fmc_flag_clear is shown as below:

Table 3-419. Function fmc_flag_clear

Function name	fmc_flag_clear
Function prototype	void fmc_flag_clear(uint32_t flag);
Function descriptions	clear the FMC flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	clear FMC flag
FMC_FLAG_BUSY	FMC busy flag bit
FMC_FLAG_RDDERR	FMC read D-bus protection error flag bit
FMC_FLAG_PGSERR	FMC program sequence error flag bit
FMC_FLAG_PGMERR	FMC program size not match error flag bit
FMC_FLAG_WPERR	FMC Erase/Program protection error flag bit
FMC_FLAG_OPERR	FMC operation error flag bit
FMC_FLAG_END	FMC end of operation flag bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the FMC_FLAG_END flag */
```

```
fmc_flag_clear(FMC_FLAG_END);
```

fmc_interrupt_enable

The description of fmc_interrupt_enable is shown as below:

Table 3-420. Function fmc_interrupt_enable

Function name	fmc_interrupt_enable
Function prototype	void fmc_interrupt_enable(uint32_t interrupt);
Function descriptions	enable FMC interrupt

Precondition	-
The called functions	-
Input parameter{in}	
interrupt	the FMC interrupt source
<i>FMC_INT_END</i>	enable FMC end of program interrupt
<i>FMC_INT_ERR</i>	enable FMC error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable FMC end of program interrupt */

fmc_unlock( );

fmc_interrupt_enable(FMC_INT_END);

fmc_lock( );
```

fmc_interrupt_disable

The description of fmc_interrupt_disable is shown as below:

Table 3-421. Function fmc_interrupt_disable

Function name	fmc_interrupt_disable
Function prototype	void fmc_interrupt_disable(uint32_t interrupt);
Function descriptions	disable FMC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	the FMC interrupt source
<i>FMC_INT_END</i>	disable FMC end of program interrupt
<i>FMC_INT_ERR</i>	disable FMC error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable FMC interrupt */

fmc_interrupt_disable(FMC_INT_END);
```

fmc_interrupt_flag_get

The description of fmc_interrupt_flag_get is shown as below:

Table 3-422. Function fmc_interrupt_flag_get

Function name	fmc_interrupt_flag_get
Function prototype	FlagStatus fmc_interrupt_flag_get(uint32_t int_flag);
Function descriptions	check interrupt flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	check FMC flag
<i>FMC_INT_FLAG_RDD ERR</i>	FMC read D/bus protection error interrupt flag bit
<i>FMC_INT_FLAG_PGS ERR</i>	FMC program sequence error interrupt flag bit
<i>FMC_INT_FLAG_PGM ERR</i>	FMC program size not match error interrupt flag bit
<i>FMC_INT_FLAG_WPE RR</i>	FMC Erase/Program protection error interrupt flag bit
<i>FMC_INT_FLAG_OPE RR</i>	FMC operation error interrupt flag bit
<i>FMC_INT_FLAG_END</i>	FMC end of operation interrupt flag bit
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check the FMC_INT_FLAG_END flag set or not*/
FlagStatus int_flag = RESET;
int_flag = fmc_interrupt_flag_get(FMC_INT_FLAG_END);
```

fmc_interrupt_flag_clear

The description of fmc_interrupt_flag_clear is shown as below:

Table 3-423. Function fmc_interrupt_flag_clear

Function name	fmc_interrupt_flag_clear
Function prototype	void fmc_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear the FMC interrupt flag
Precondition	-
The called functions	-

Input parameter{in}	
int_flag	clear FMC interrupt flag
FMC_INT_FLAG_RDD_ERR	FMC read D/bus protection error interrupt flag bit
FMC_INT_FLAG_PGS_ERR	FMC program sequence error interrupt flag bit
FMC_INT_FLAG_PGM_ERR	FMC program size not match error interrupt flag bit
FMC_INT_FLAG_WPE_RR	FMC Erase/Program protection error interrupt flag bit
FMC_INT_FLAG_OPE_RR	FMC operation error interrupt flag bit
FMC_INT_FLAG_END	FMC end of operation interrupt flag bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* clear the FMC_INT_FLAG_END flag */

fmc_interrupt_flag_clear(FMC_INT_FLAG_END);

```

fmc_state_get

The description of fmc_state_get is shown as below:

Table 3-424. Function fmc_state_get

Function name	fmc_state_get
Function prototype	fmc_state_enum fmc_state_get(void);
Function descriptions	get the FMC state
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to fmc_state_enum

Example:

```

/* get the FMC state */

fmc_state_enum state = fmc_state_get();

```

fmc_ready_wait

The description of fmc_ready_wait is shown as below:

Table 3-425. Function fmc_ready_wait

Function name	fmc_ready_wait
Function prototype	fmc_state_enum fmc_ready_wait(void);
Function descriptions	check whether FMC is ready or not
Precondition	-
The called functions	fmc_state_get()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to fmc state enum

Example:

```
/* check whether FMC is ready or not */
fmc_state_enum state = fmc_ready_wait ( );
```

3.14. FWDGT

The free watchdog timer (FWDGT) is a hardware timing circuitry that can be used to detect system failures due to software malfunctions. It's suitable for the situation that requires an independent environment and lower timing accuracy. The FWDGT registers are listed in chapter [3.14.1](#) the FWDGT firmware functions are introduced in chapter [3.14.2](#).

3.14.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

Table 3-426. FWDGT Registers

Registers	Descriptions
FWDGT_CTL	Control register
FWDGT_PSC	Prescaler register
FWDGT_RLD	Reload register
FWDGT_STAT	Status register

3.14.2. Descriptions of Peripheral functions

FWDGT firmware functions are listed in the table shown as below:

Table 3-427. FWDGT firmware function

Function name	Function description
fwdgt_write_enable	enable write access to FWDGT_PSC and FWDGT_RLD
fwdgt_write_disable	disable write access to FWDGT_PSC and FWDGT_RLD
fwdgt_enable	start the free watchdog timer counter
fwdgt_prescaler_value_config	configure the free watchdog timer counter prescaler value
fwdgt_reload_value_config	configure the free watchdog timer counter reload value
fwdgt_counter_reload	reload the counter of FWDGT
fwdgt_config	configure counter reload value, and prescaler divider value
fwdgt_flag_get	get flag state of FWDGT

fwdgt_write_enable

The description of fwdgt_write_enable is shown as below:

Table 3-428. Function fwdgt_write_enable

Function name	fwdgt_write_enable
Function prototype	void fwdgt_write_enable(void);
Function descriptions	enable write access to FWDGT_PSC and FWDGT_RLD
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable write access to FWDGT_PSC and FWDGT_RLD */
```

```
fwdgt_write_enable( );
```

fwdgt_write_disable

The description of fwdgt_write_disable is shown as below:

Table 3-429. Function fwdgt_write_disable

Function name	fwdgt_write_disable
Function prototype	void fwdgt_write_disable(void);
Function descriptions	disable write access to FWDGT_PSC and FWDGT_RLD
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable write access to FWDGT_PSC and FWDGT_RLD */
fwdgt_write_disable( );
```

fwdgt_enable

The description of fwdgt_enable is shown as below:

Table 3-430. Function fwdgt_enable

Function name	fwdgt_enable
Function prototype	void fwdgt_enable(void);
Function descriptions	start the free watchdog timer counter
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* start the free watchdog timer counter */
fwdgt_enable ( );
```

fwdgt_prescaler_value_config

The description of fwdgt_prescaler_value_config is shown as below:

Table 3-431. Function fwdgt_prescaler_value_config

Function name	fwdgt_prescaler_value_config
Function prototype	ErrStatus fwdgt_prescaler_value_config(uint16_t prescaler_value);
Function descriptions	configure the free watchdog timer counter prescaler value
Precondition	-
Input parameter{in}	
prescaler_value	specify prescaler value
<i>FWDGT_PSC_DIV4</i>	FWDGT prescaler set to 4
<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16
<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32
<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64
<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR / SUCCESS

Example:

```
/* set FWDGT prescaler to 256 */
ErrStatus flag;
flag = fwdgt_prescaler_value_config (FWDGT_PSC_DIV256);
```

fwdgt_reload_value_config

The description of fwdgt_reload_value_config is shown as below:

Table 3-432. Function fwdgt_reload_value_config

Function name	fwdgt_reload_value_config
Function prototype	ErrStatus fwdgt_reload_value_config(uint16_t reload_value);
Function descriptions	configure the free watchdog timer counter reload value
Precondition	-
Input parameter{in}	
reload_value	reload_value: specify window value(0x0000 - 0x0FFF)
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR / SUCCESS

Example:

```
/* set FWDGT reload value to 0x0FFF */

ErrStatus flag;

flag = fwdgt_reload_value_config (0x0FFF);
```

fwdgt_counter_reload

The description of fwdgt_counter_reload is shown as below:

Table 3-433. Function fwdgt_counter_reload

Function name	fwdgt_counter_reload
Function prototype	void fwdgt_counter_reload(void);
Function descriptions	reload the counter of FWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reload FWDGT counter */

fwdgt_counter_reload ( );
```

fwdgt_config

The description of fwdgt_config is shown as below:

Table 3-434. Function fwdgt_config

Function name	fwdgt_config
Function prototype	ErrStatus fwdgt_config(uint16_t reload_value, uint8_t prescaler_div);
Function descriptions	configure counter reload value, and prescaler divider value
Precondition	-
The called functions	-
Input parameter{in}	
reload_value	specify reload value(0x0000 - 0x0FFF)-
Input parameter{in}	
prescaler_div	FWDGT prescaler value-
<i>FWDGT_PSC_DIV4</i>	FWDGT prescaler set to 4
<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16
<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32
<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64
<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* configure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

fwdgt_flag_get

The description of fwdgt_flag_get is shown as below:

Table 3-435. Function fwdgt_flag_get fwdgt_write_disable

Function name	fwdgt_flag_get
Function prototype	FlagStatus fwdgt_flag_get(uint16_t flag);
Function descriptions	get flag state of FWDGT
Precondition	-
The called functions	-
Input parameter{in}	
flag	flag to get
<i>FWDGT_FLAG_PUD</i>	a write operation to FWDGT_PSC register is on going
<i>FWDGT_FLAG_RUD</i>	a write operation to FWDGT_RLD register is on going
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```

/* test if a prescaler value update is on going */

FlagStatus status;

status = fwdgt_flag_get (FWDGT_FLAG_PUD);

if(status == RESET)

{
    ...
}

}else

{
    ...
}

```

3.15. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.15.1](#), the GPIO firmware functions are introduced in chapter [错误!未找到引用源。](#).

3.15.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

Table 3-436. GPIO Registers

Registers	Descriptions
GPIOx_CTL	GPIO port control register
GPIO_OMODE	GPIO port output mode register
GPIO_OSPD	GPIO port output speed register
GPIO_PUD	GPIO port pull-up/pull-down register
GPIO_ISTAT	GPIO port input status register
GPIO_OCTL	GPIO port output control register
GPIO_BOP	GPIO port bit operate register
GPIO_LOCK	GPIO port configuration lock register
GPIO_AFSEL0	GPIO alternate function selected register 0
GPIO_AFSEL1	GPIO alternate function selected register 1
GPIO_BC	GPIO bit clear register
GPIO_TG	GPIO port bit toggle register

3.15.2. Descriptions of Peripheral functions

GPIO firmware functions are listed in the table shown as below:

Table 3-437. GPIO firmware function

Function name	Function description
gpio_deinit	reset GPIO port
gpio_mode_set	set GPIO mode
gpio_output_options_set	set GPIO output type and speed
gpio_bit_set	set GPIO pin bit
gpio_bit_reset	reset GPIO pin bit
gpio_bit_write	write data to the specified GPIO pin
gpio_port_write	write data to the specified GPIO port
gpio_input_bit_get	get GPIO pin input status
gpio_input_port_get	get GPIO port input status
gpio_output_bit_get	get GPIO pin output status
gpio_output_port_get	get GPIO port output status
gpio_af_set	set GPIO alternate function
gpio_pin_lock	lock GPIO pin bit
gpio_bit_toggle	toggle GPIO pin status
gpio_port_toggle	toggle GPIO port status

gpio_deinit

The description of gpio_deinit is shown as below:

Table 3-438. Function gpio_deinit

Function name	gpio_deinit
Function prototype	void gpio_deinit(uint32_t gpio_periph);
Function descriptions	reset GPIO port
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset GPIOA */
gpio_deinit(GPIOA);
```

gpio_mode_set

The description of gpio_mode_set is shown as below:

Table 3-439. Function gpio_mode_set

Function name	gpio_mode_set
Function prototype	void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin);
Function descriptions	set GPIO mode
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
mode	GPIO pin mode
GPIO_MODE_INPUT	input mode
GPIO_MODE_OUTPUT	output mode
GPIO_MODE_AF	alternate function mode
GPIO_MODE_ANALOG	analog mode
Input parameter{in}	
pull_up_down	GPIO pin with pull-up or pull-down resistor

<i>GPIO_PUPD_NONE</i>	floating mode, no pull-up and pull-down resistors
<i>GPIO_PUPD_PULLUP</i>	with pull-up resistor
<i>GPIO_PUPD_PULLDOWN</i>	with pull-down resistor
Input parameter{in}	
pin	GPIO pin
<i>GPIO_PIN_x</i>	<i>GPIO_PIN_x</i> (x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set PA0 with pull-up input mode */
gpio_mode_set(GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_PULLUP, GPIO_PIN_0);
```

gpio_output_options_set

The description of `gpio_output_options_set` is shown as below:

Table 3-440. Function `gpio_output_options_set`

Function name	<code>gpio_output_options_set</code>
Function prototype	<code>void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);</code>
Function descriptions	set GPIO output type and speed
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	<i>GPIOx</i> (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
otype	GPIO pin output mode
<i>GPIO_OTYPE_PP</i>	push pull mode
<i>GPIO_OTYPE_OD</i>	open drain mode
Input parameter{in}	
speed	gpio output max speed value
<i>GPIO_OSPEED_2MHZ</i>	output max speed 2MHz
<i>GPIO_OSPEED_25MHZ</i>	output max speed 25MHz

<code>GPIO_OSPEED_50MHZ</code>	output max speed 50MHz
<code>GPIO_OSPEED_MAX</code>	output max speed more than 50MHz
Input parameter{in}	
<code>pin</code>	GPIO pin
<code>GPIO_PIN_x</code>	<code>GPIO_PIN_x</code> ($x=0..15$)
<code>GPIO_PIN_ALL</code>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure PA0 as push pull mode */
gpio_output_options_set(GPIOA, GPIO_OTYPE_PP, GPIO_OSPEED_2MHZ,
GPIO_PIN_0);
```

gpio_bit_set

The description of `gpio_bit_set` is shown as below:

Table 3-441. Function `gpio_bit_set`

Function name	<code>gpio_bit_set</code>
Function prototype	<code>void gpio_bit_set(uint32_t gpio_periph, uint32_t pin);</code>
Function descriptions	set GPIO pin bit
Precondition	-
The called functions	-
Input parameter{in}	
<code>gpio_periph</code>	GPIO port
<code>GPIOx</code>	<code>GPIOx</code> ($x = A,B,C,D,E,F,G,H,I$)
Input parameter{in}	
<code>pin</code>	GPIO pin
<code>GPIO_PIN_x</code>	<code>GPIO_PIN_x</code> ($x=0..15$)
<code>GPIO_PIN_ALL</code>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set PA0*/
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

gpio_bit_reset

The description of gpio_bit_reset is shown as below:

Table 3-442. Function gpio_bit_reset

Function name	gpio_bit_reset
Function prototype	void gpio_bit_reset(uint32_t gpio_periph,uint32_t pin);
Function descriptions	reset GPIO pin bit
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset PA0 */
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

gpio_bit_write

The description of gpio_bit_write is shown as below:

Table 3-443. Function gpio_bit_write

Function name	gpio_bit_write
Function prototype	void gpio_bit_write(uint32_t gpio_periph, uint32_t pin, bit_status bit_value);
Function descriptions	write data to the specified GPIO pin
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins

Input parameter{in}	
bit_value	SET or RESET
<i>RESET</i>	clear the port pin
<i>SET</i>	set the port pin
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write 1 to PA0 */
gpio_bit_write(GPIOA, GPIO_PIN_0, SET);
```

gpio_port_write

The description of `gpio_port_write` is shown as below:

Table 3-444. Function `gpio_port_write`

Function name	gpio_port_write
Function prototype	void gpio_port_write(uint32_t gpio_periph, uint16_t data);
Function descriptions	write data to the specified GPIO port
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
data	specify the value to be written to the port output data register
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write 1010 0101 1010 0101 to Port A */
gpio_port_write(GPIOA, 0xA5A5);
```

gpio_input_bit_get

The description of `gpio_input_bit_get` is shown as below:

Table 3-445. Function `gpio_input_bit_get`

Function name	gpio_input_bit_get
----------------------	--------------------

Function prototype	FlagStatus gpio_input_bit_get(uint32_t gpio_periph, uint32_t pin);
Function descriptions	get GPIO pin input status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get status of PA0 */

FlagStatus bit_state;

bit_state = gpio_input_bit_get (GPIOA, GPIO_PIN_0);

gpio_input_port_get
```

The description of `gpio_input_port_get` is shown as below:

Table 3-446. Function `gpio_input_port_get`

Function name	gpio_input_port_get
Function prototype	uint16_t gpio_input_port_get(uint32_t gpio_periph);
Function descriptions	get GPIO port input status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Output parameter{out}	
-	-
Return value	
uint16_t	0x0000-0xFFFF

Example:

```
/* get input value of Port A */

uint16_t port_state;
```

```
port_state = gpio_input_bit_get(GPIOA);
```

gpio_output_bit_get

The description of gpio_output_bit_get is shown as below:

Table 3-447. Function gpio_output_bit_get

Function name	gpio_output_bit_get
Function prototype	FlagStatus gpio_output_bit_get(uint32_t gpio_periph, uint32_t pin);
Function descriptions	get GPIO pin output status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get output status of PA0 */
FlagStatus bit_state;
bit_state = gpio_output_bit_get(GPIOA, GPIO_PIN_0);
```

gpio_output_port_get

The description of gpio_output_port_get is shown as below:

Table 3-448. Function gpio_output_port_get

Function name	gpio_output_port_get
Function prototype	uint16_t gpio_output_port_get(uint32_t gpio_periph);
Function descriptions	get GPIO port output status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Output parameter{out}	

-	-
Return value	
Uint16_t	0x0000-0xFFFF

Example:

```
/* get output value of Port A */

uint16_t port_state;

port_state = gpio_output_port_get(GPIOA);
```

gpio_af_set

The description of `gpio_af_set` is shown as below:

Table 3-449. Function `gpio_pin_remap_config`

Function name	gpio_af_set
Function prototype	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);
Function descriptions	set GPIO alternate function
Precondition	-
The called functions	-
Input parameter{in}	
<code>gpio_periph</code>	GPIO port
<code>GPIOx</code>	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
<code>alt_func_num</code>	GPIO pin af function
<code>GPIO_AF_0</code>	SYSTEM
<code>GPIO_AF_1</code>	TIMER0, TIMER1
<code>GPIO_AF_2</code>	TIMER2, TIMER3, TIMER4
<code>GPIO_AF_3</code>	TIMER7, TIMER8, TIMER9, TIMER10
<code>GPIO_AF_4</code>	I2C0, I2C1, I2C2
<code>GPIO_AF_5</code>	SPI0, SPI1, SPI2, SPI3, SPI4, SPI5
<code>GPIO_AF_6</code>	SPI2, SPI3, SPI4
<code>GPIO_AF_7</code>	USART0, USART1, USART2, SPI1, SPI2
<code>GPIO_AF_8</code>	UART3, UART4, USART5, UART6, USART7
<code>GPIO_AF_9</code>	CAN0, CAN1, TLI, TIMER11, TIMER12, TIMER13, I2C1, I2C2, CTC
<code>GPIO_AF_10</code>	USB_FS, USB_HS
<code>GPIO_AF_11</code>	ENET
<code>GPIO_AF_12</code>	EXMC, SDIO, USB_HS
<code>GPIO_AF_13</code>	DCI
<code>GPIO_AF_14</code>	TLI
<code>GPIO_AF_15</code>	EVENTOUT
Input parameter{in}	
<code>pin</code>	GPIO pin

<i>GPIO_PIN_x</i>	GPIO_PIN_x (x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*set PA0 alternate function 0*/
gpio_af_set(GPIOA, GPIO_AF_0, GPIO_PIN_0);
```

gpio_pin_lock

The description of `gpio_pin_lock` is shown as below:

Table 3-450. Function `gpio_pin_lock`

Function name	gpio_pin_lock
Function prototype	void gpio_pin_lock(uint32_t gpio_periph,uint32_t pin);
Function descriptions	lock GPIO pin bit
Precondition	-
The called functions	-
Input parameter{in}	
<code>gpio_periph</code>	GPIO port
<code>GPIOx</code>	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
<code>pin</code>	GPIO pin
<code>GPIO_PIN_x</code>	GPIO_PIN_x (x=0..15)
<code>GPIO_PIN_ALL</code>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock PA0 */
gpio_pin_lock (GPIOA, GPIO_PIN_0);
```

gpio_bit_toggle

The description of `gpio_bit_toggle` is shown as below:

Table 3-451. Function `gpio_bit_toggle`

Function name	gpio_bit_toggle
Function prototype	void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin);

Function descriptions	toggle GPIO pin status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* toggle PA0 */

gpio_bit_toggle (GPIOA, GPIO_PIN_0);
```

gpio_port_toggle

The description of **gpio_port_toggle** is shown as below:

Table 3-452. Function gpio_port_toggle

Function name	gpio_port_toggle
Function prototype	void gpio_port_toggle(uint32_t gpio_periph);
Function descriptions	toggle GPIO port status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* toggle GPIOA */
```

```
gpio_port_toggle(GPIOA);
```

3.16. I2C

The I2C (inter-integrated circuit) module provides an I2C interface which is an industry standard two-line serial interface for MCU to communicate with external I2C interface. The I2C registers are listed in chapter [3.16.1](#), the I2C firmware functions are introduced in chapter [3.16.2](#).

3.16.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

Table 3-453. I2C Registers

Registers	Descriptions
I2C_CTL0	Control register 0
I2C_CTL1	Control register 1
I2C_SADDR0	Slave address register 0
I2C_SADDR1	Slave address register 1
I2C_DATA	Transfer buffer register
I2C_STAT0	Transfer status register 0
I2C_STAT1	Transfer status register 1
I2C_CKCFG	Clock configure register
I2C_RT	Rise time register
I2C_FCTL	Filter control register
I2C_SAMCS	SAM control and status register

3.16.2. Descriptions of Peripheral functions

I2C firmware functions are listed in the table shown as below:

Table 3-454. I2C firmware function

Function name	Function description
i2c_deinit	reset I2C
i2c_clock_config	configure I2C clock

Function name	Function description
i2c_mode_addr_config	configure I2C address
i2c_smbus_type_config	SMBus type selection
i2c_ack_config	whether or not to send an ACK
i2c_ackpos_config	configure I2C POAP position
i2c_master_addressing	master send slave address
i2c_dualaddr_enable	dual-address mode switch
i2c_enable	enable I2C
i2c_disable	disable I2C
i2c_start_on_bus	generate a START condition on I2C bus
i2c_stop_on_bus	generate a STOP condition on I2C bus
i2c_data_transmit	I2C transmit data function
i2c_data_receive	I2C receive data function
i2c_dma_config	configure I2C DMA mode
i2c_dma_last_transfer_config	configure whether next DMA EOT is DMA last transfer or not
i2c_stretch_scl_low_config	whether to stretch SCL low when data is not ready in slave mode
i2c_slave_response_to_gcall_config	whether or not to response to a general call
i2c_software_reset_config	configure software reset of I2C
i2c_pec_config	configure I2C PEC calculation
i2c_pec_transfer_config	configure whether to transfer PEC value
i2c_pec_value_get	packet error checking value
i2c_smbus_alert_config	configure I2C alert through SMBA pin
i2c_smbus_arp_config	configure I2C ARP protocol in SMBus
i2c_analog_noise_filter_disable	I2C analog noise filter disable
i2c_analog_noise_filter_enable	I2C analog noise filter enable
i2c_digital_noise_filter_config	digital noise filter
i2c_sam_enable	enable SAM_V interface

Function name	Function description
i2c_sam_disable	disable SAM_V interface
i2c_sam_timeout_enable	enable SAM_V interface timeout detect
i2c_sam_timeout_disable	disable SAM_V interface timeout detect
i2c_flag_get	get I2C flag status
i2c_flag_clear	clear I2C flag status
i2c_interrupt_enable	enable I2C interrupt
i2c_interrupt_disable	disable I2C interrupt
i2c_interrupt_flag_get	get I2C interrupt flag status
i2c_interrupt_flag_clear	clear I2C interrupt flag status

i2c_deinit

The description of i2c_deinit is shown as below:

Table 3-455. Function i2c_deinit

Function name	i2c_deinit
Function prototype	void i2c_deinit(uint32_t i2c_periph);
Function descriptions	reset I2C
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset I2C0 */
```

```
i2c_deinit(I2C0);
```

i2c_clock_config

The description of i2c_clock_config is shown as below:

Table 3-456. Function i2c_clock_config

Function name	i2c_clock_config
Function prototype	void i2c_clock_config(uint32_t i2c_periph, uint32_t clkspeed, uint32_t dutycyc);
Function descriptions	I2C clock configure
Precondition	-
The called functions	rcu_clock_freq_get
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
clkspeed	i2c clock speed
Input parameter{in}	
dutycyc	duty cycle in fast mode
I2C_DTCY_2	T_low/T_high=2
I2C_DTCY_16_9	T_low/T_high=16/9
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure I2C0 clock speed as 100KHz*/
i2c_clock_config(I2C0, 100000, I2C_DTCY_2);
```

i2c_mode_addr_config

The description of i2c_mode_addr_config is shown as below:

Table 3-457. Function i2c_mode_addr_config

Function name	i2c_mode_addr_config
Function prototype	void i2c_mode_addr_config(uint32_t i2c_periph, uint32_t mode, uint32_t addformat, uint32_t addr);
Function descriptions	configure I2C address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
mode	I2C mode select
<i>I2C_I2CMODE_ENABLE</i>	I2C mode
<i>I2C_SMBUSMODE_ENABLE</i>	SMBus mode
Input parameter{in}	
addformat	7bits or 10bits
<i>I2C_ADDFORMAT_7BITS</i>	address format is 7 bits
<i>I2C_ADDFORMAT_10BITS</i>	address format is 10 bits
Input parameter{in}	
addr	I2C address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure I2C0 address as 0x82, using 7 bits */
```

```
i2c_mode_addr_config(I2C0, I2C_I2CMODE_ENABLE, I2C_ADDFORMAT_7BITS, 0x82);
```

i2c_smbus_type_config

The description of i2c_smbus_type_config is shown as below:

Table 3-458. Function i2c_smbus_type_config

Function name	i2c_smbus_type_config
Function prototype	void i2c_smbus_type_config(uint32_t i2c_periph, uint32_t type);
Function descriptions	SMBus type selection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
type	Device or host
I2C_SMBUS_DEVICE	device
I2C_SMBUS_HOST	host
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config I2C0 as SMBUS host type*/
i2c_smbus_type_config (I2C0, I2C_SMBUS_HOST);
```

i2c_ack_config

The description of i2c_ack_config is shown as below:

Table 3-459. Function i2c_ack_config

Function name	i2c_ack_config
Function prototype	void i2c_ack_config(uint32_t i2c_periph, uint32_t ack);

Function descriptions	whether or not to send an ACK
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
ack	I2C peripheral
<i>I2C_ACK_ENABLE</i>	ACK will be sent
<i>I2C_ACK_DISABLE</i>	ACK will not be sent
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 will send ACK */
i2c_ack_config (I2C0, I2C_ACK_ENABLE);
```

i2c_ackpos_config

The description of i2c_ackpos_config is shown as below:

Table 3-460. Function i2c_ackpos_config

Function name	i2c_ackpos_config
Function prototype	void i2c_ackpos_config(uint32_t i2c_periph, uint32_t pos);
Function descriptions	I2C POAP position configure
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)

Input parameter{in}	
pos	ACK position
<i>I2C_ACKPOS_CURRENT</i>	whether to send ACK or not for the current
<i>I2C_ACKPOS_NEXT</i>	whether to send ACK or not for the next byte
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* The ACK of I2C0 is send for the current frame*/
i2c_ackpos_config (I2C0, I2C_ACKPOS_CURRENT);
```

i2c_master_addressing

The description of i2c_master_addressing is shown as below:

Table 3-461. Function i2c_master_addressing

Function name	i2c_master_addressing
Function prototype	void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection);
Function descriptions	void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection);
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
addr	slave address
Input parameter{in}	
trandirection	transmitter or receiver

<i>I2C_TRANSMITTER</i>	transmitter
<i>I2C_RECEIVER</i>	receiver
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* send slave address to I2C bus and I2C0 act as receiver */

i2c_master_addressing(I2C0, 0x82, I2C_RECEIVER);
```

i2c_dualaddr_enable

The description of i2c_dualaddr_enable is shown as below:

Table 3-462. Function i2c_dualaddr_enable

Function name	i2c_dualaddr_enable
Function prototype	void i2c_dualaddr_enable(uint32_t i2c_periph, uint32_t dualaddr);
Function descriptions	dual-address mode switch
Precondition	-
The called functions	-
Input parameter{in}	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
<i>dualaddr</i>	Enable or disable
<i>I2C_DUADEN_DISABLE</i>	disable dual-address mode
<i>I2C_DUADEN_ENABLE</i>	enable dual-address mode
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable I2C0 dual-address*/
i2c_dualaddr_enable (I2C0, I2C_DUADEN_ENABLE);
```

i2c_enable

The description of i2c_enable is shown as below:

Table 3-463. Function i2c_enable

Function name	i2c_enable
Function prototype	void i2c_enable(uint32_t i2c_periph);
Function descriptions	enable I2C
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 */
i2c_enable (I2C0);
```

i2c_disable

The description of i2c_disable is shown as below:

Table 3-464. Function i2c_disable

Function name	i2c_disable
Function prototype	void i2c_disable(uint32_t i2c_periph);
Function descriptions	disable I2C

Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 */
i2c_disable (I2C0);
```

i2c_start_on_bus

The description of i2c_start_on_bus is shown as below:

Table 3-465. Function i2c_start_on_bus

Function name	i2c_start_on_bus
Function prototype	void i2c_start_on_bus(uint32_t i2c_periph);
Function descriptions	generate a START condition on I2C bus
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 send a start condition to I2C bus */
```

```
i2c_start_on_bus (I2C0);
```

i2c_stop_on_bus

The description of i2c_stop_on_bus is shown as below:

Table 3-466. Function i2c_stop_on_bus

Function name	i2c_stop_on_bus
Function prototype	void i2c_stop_on_bus(uint32_t i2c_periph);
Function descriptions	generate a STOP condition on I2C bus
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 generate a STOP condition to I2C bus */
```

```
i2c_stop_on_bus (I2C0);
```

i2c_data_transmit

The description of i2c_data_transmit is shown as below:

Table 3-467. Function i2c_data_transmit

Function name	i2c_data_transmit
Function prototype	void i2c_data_transmit(uint32_t i2c_periph, uint8_t data);
Function descriptions	I2C transmit data function
Precondition	-
The called functions	-

Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
data	transmit data
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 transmit data */

i2c_data_transmit (I2C0);
```

i2c_data_receive

The description of i2c_data_receive is shown as below:

Table 3-468. Function i2c_data_receive

Function name	i2c_data_receive
Function prototype	uint8_t i2c_data_receive(uint32_t i2c_periph);
Function descriptions	I2C receive data function
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
uint8_t	0x00..0xFF

Example:

```
/* I2C0 receive data */

uint8_t i2c_receiver;
i2c_receiver = i2c_data_receive(I2C0);
```

i2c_dma_config

The description of i2c_dma_config is shown as below:

Table 3-469. Function i2c_dma_config

Function name	i2c_dma_config
Function prototype	void i2c_dma_config(uint32_t i2c_periph, uint32_t dmastate);
Function descriptions	configure I2C DMA mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
dmastate	On or off
<i>I2C_DMA_ON</i>	DMA mode enable
<i>I2C_DMA_OFF</i>	DMA mode disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 DMA mode enable */

i2c_dma_config(I2C0, I2C_DMA_ON);
```

i2c_dma_last_transfer_enable

The description of i2c_dma_last_transfer_enable is shown as below:

Table 3-470. Function i2c_dma_last_transfer_enable

Function name	i2c_dma_last_transfer_enable
Function prototype	void i2c_dma_last_transfer_enable(uint32_t i2c_periph, uint32_t dmalast);
Function descriptions	flag indicating DMA last transfer
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
dmalast	next DMA EOT is the last transfer or not
<i>I2C_DMALST_ON</i>	next DMA EOT is the last transfer
<i>I2C_DMALST_OFF</i>	next DMA EOT is not the last transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* next DMA EOT is the last transfer */

i2c_dma_last_transfer_enable (I2C0, I2C_DMALST_ON);

```

i2c_stretch_scl_low_config

The description of i2c_stretch_scl_low_config is shown as below:

Table 3-471. Function i2c_stretch_scl_low_config

Function name	i2c_stretch_scl_low_config
Function prototype	void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara);
Function descriptions	whether to stretch SCL low when data is not ready in slave mode
Precondition	-
The called functions	-

Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
stretchpara	SCL stretching enable or disable
I2C_SCLSTRETCH_ENABLE	enable SCL stretching
I2C_SCLSTRETCH_DISABLE	enable SCL stretching
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* stretch SCL low when data is not ready in slave mode */
i2c_stretch_scl_low_config (I2C0, I2C_SCLSTRETCH_ENABLE);
```

i2c_slave_response_to_gcall_config

The description of i2c_slave_response_to_gcall_config is shown as below:

Table 3-472. Function i2c_slave_response_to_gcall_config

Function name	i2c_slave_response_to_gcall_config
Function prototype	void i2c_slave_response_to_gcall_config(uint32_t i2c_periph, uint32_t gcallpara);
Function descriptions	whether or not to response to a general call
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	

gcallpara	response to a general call or not
<i>I2C_GCEN_ENABLE</i>	slave will response to a general call
<i>I2C_GCEN_DISABLE</i>	slave will not response to a general call
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* I2C0 will response to a general call */

i2c_slave_response_to_gcall_config (I2C0, I2C_GCEN_ENABLE);

```

i2c_software_reset_config

The description of i2c_software_reset_config is shown as below:

Table 3-473. Function i2c_software_reset_config

Function name	i2c_software_reset_config
Function prototype	void i2c_software_reset_config(uint32_t i2c_periph, uint32_t sreset);
Function descriptions	configure software reset of I2C
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
sreset	under reset or not
<i>I2C_SRESET_SET</i>	I2C is under reset
<i>I2C_SRESET_RESET</i>	I2C is not under reset
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* software reset I2C0 */

i2c_software_reset_config (I2C0, I2C_SRESET_SET);
```

i2c_pec_config

The description of i2c_pec_config is shown as below:

Table 3-474. Function i2c_pec_config

Function name	i2c_pec_config
Function prototype	void i2c_pec_config (uint32_t i2c_periph, uint32_t pecstate);
Function descriptions	configure I2C PEC calculation
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
pecstate	On or off
I2C_PEC_ENABLE	PEC calculation on
I2C_PEC_DISABLE	PEC calculation off
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* Enable I2C PEC calculation */

i2c_pec_config (I2C0, I2C_PEC_ENABLE);
```

i2c_pec_transfer_config

The description of i2c_pec_transfer_config is shown as below:

Table 3-475. Function i2c_pec_transfer_config

Function name	i2c_pec_transfer_config
Function prototype	void i2c_pec_transfer_config(uint32_t i2c_periph, uint32_t pecpara);
Function descriptions	configure whether to transfer PEC value
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
pecpara	Transfer PEC or not
<i>I2C_PECTRANS_ENA</i> <i>BLE</i>	transfer PEC
<i>I2C_PECTRANS_DISA</i> <i>BLE</i>	not transfer PEC
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 transfer PEC */

i2c_pec_transfer_config(I2C0, I2C_PECTRANS_ENABLE);
```

i2c_pec_value_get

The description of i2c_pec_value_get is shown as below:

Table 3-476. Function i2c_pec_value_get

Function name	i2c_pec_value_get
Function prototype	uint8_t i2c_pec_value_get(uint32_t i2c_periph);
Function descriptions	get packet error checking value
Precondition	-

The called functions	-
Input parameter{in}	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
<i>uint8_t</i>	0..255

Example:

```
/* I2C0 get packet error checking value */
uint8_t pec_value;
pec_value = i2c_pec_value_get (I2C0);
```

i2c_smbus_alert_config

The description of *i2c_smbus_alert_config* is shown as below:

Table 3-477. Function i2c_smbus_alert_config

Function name	i2c_smbus_alert_config
Function prototype	void i2c_smbus_alert_config(uint32_t i2c_periph, uint32_t smbuspara);
Function descriptions	configure I2C alert through SMBA pin
Precondition	-
The called functions	-
Input parameter{in}	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
<i>smbuspara</i>	issue alert through SMBA pin or not
<i>I2C_SALTSEND_ENAB LE</i>	issue alert through SMBA pin
<i>I2C_SALTSEND_DISA BLE</i>	not issue alert through SMBA pin

Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* I2C0 issue alert through SMBA pin enable*/
i2c_smbus_alert_config(I2C0, I2C_SALTSEND_ENABLE);

```

i2c_smbus_arp_config

The description of i2c_smbus_arp_config is shown as below:

Table 3-478. Function i2c_smbus_arp_config

Function name	i2c_smbus_arp_config
Function prototype	void i2c_smbus_arp_config(uint32_t i2c_periph, uint32_t arpstate);
Function descriptions	configure I2C ARP protocol in SMBus
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
arpstate	ARP protocol in SMBus switch
I2C_ARP_ENABLE	enable ARP
I2C_ARP_DISABLE	disable ARP
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable I2C0 ARP protocol in SMBus switch */

```

```
i2c_smbus_arp_config(I2C0, I2C_ARP_ENABLE);
```

i2c_analog_noise_filter_disable

The description of i2c_analog_noise_filter_disable is shown as below:

Table 3-479. Function i2c_analog_noise_filter_disable

Function name	i2c_analog_noise_filter_disable
Function prototype	void i2c_analog_noise_filter_disable(uint32_t i2c_periph);
Function descriptions	I2C analog noise filter disable
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 analog noise filter */
i2c_analog_noise_filter_disable (I2C0);
```

i2c_analog_noise_filter_enable

The description of i2c_analog_noise_filter_enable is shown as below:

Table 3-480. Function i2c_analog_noise_filter_enable

Function name	i2c_analog_noise_filter_enable
Function prototype	void i2c_analog_noise_filter_enable(uint32_t i2c_periph);
Function descriptions	I2C analog noise filter enable
Precondition	-
The called functions	-
Input parameter{in}	

i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 analog noise filter */
i2c_analog_noise_filter_enable (I2C0);
```

i2c_digital_noise_filter_config

The description of i2c_digital_noise_filter_config is shown as below:

Table 3-481. Function i2c_digital_noise_filter_config

Function name	i2c_digital_noise_filter_config
Function prototype	void i2c_digital_noise_filter_config(uint32_t i2c_periph, i2c_digital_filter_enum dfilterpara);
Function descriptions	config digital noise filter
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
dfilterpara	filtered spiker's length
<i>i2c_digital_filter_enum</i>	maximum filter spikes's length
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* config I2C0 digital noise filter as I2C_DF_1PCLK */

i2c_digital_noise_filter_config (I2C0, I2C_DF_1PCLK);

```

i2c_sam_enable

The description of i2c_sam_enable is shown as below:

Table 3-482. Function i2c_sam_enable

Function name	i2c_sam_enable
Function prototype	void i2c_sam_enable (uint32_t i2c_periph);
Function descriptions	enable SAM_V interface
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable I2C0 SAM_V interface */

i2c_sam_enable (I2C0);

```

i2c_sam_disable

The description of i2c_sam_disable is shown as below:

Table 3-483. Function i2c_sam_disable

Function name	i2c_sam_disable
Function prototype	void i2c_sam_disable (uint32_t i2c_periph);
Function descriptions	disable SAM_V interface
Precondition	-

The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 SAM_V interface*/
i2c_sam_disable (I2C0);
```

i2c_sam_timeout_enable

The description of i2c_sam_timeout_enable is shown as below:

Table 3-484. Function i2c_sam_timeout_enable

Function name	i2c_sam_timeout_enable
Function prototype	void i2c_sam_timeout_enable (uint32_t i2c_periph);
Function descriptions	enable SAM_V interface timeout detect
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 SAM_V interface timeout detect */
```

```
i2c_sam_timeout_enable (I2C0);
```

i2c_sam_timeout_disable

The description of i2c_sam_timeout_disable is shown as below:

Table 3-485. Function i2c_sam_timeout_disable

Function name	i2c_sam_timeout_disable
Function prototype	void i2c_sam_timeout_disable (uint32_t i2c_periph);
Function descriptions	disable SAM_V interface timeout detect
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 SAM_V interface timeout detect */

i2c_sam_timeout_disable (I2C0);
```

i2c_flag_get

The description of i2c_flag_get is shown as below:

Table 3-486. Function i2c_flag_get

Function name	i2c_flag_get
Function prototype	FlagStatus i2c_flag_get(uint32_t i2c_periph,uint32_t flag);
Function descriptions	get I2C flag status
Precondition	-
The called functions	-
Input parameter{in}	

i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
flag	specify get which flag
<i>I2C_FLAG_SBSEND</i>	start condition send out
<i>I2C_FLAG_ADDSEND</i>	address is sent in master mode or received and matches in slave mode
<i>I2C_FLAG_BTC</i>	byte transmission finishes
<i>I2C_FLAG_ADD10SEN_D</i>	header of 10-bit address is sent in master mode
<i>I2C_FLAG_STPDET</i>	stop condition detected in slave mode
<i>I2C_FLAG_RBNE</i>	I2C_DATA is not Empty during receiving
<i>I2C_FLAG_TBE</i>	I2C_DATA is empty during transmitting
<i>I2C_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus
<i>I2C_FLAG_LOSTARB</i>	arbitration lost in master mode
<i>I2C_FLAG_AERR</i>	acknowledge error
<i>I2C_FLAG_OVERRR</i>	overrun or underrun situation occurs in slave mode
<i>I2C_FLAG_PECERR</i>	PEC error when receiving data
<i>I2C_FLAG_SMBTO</i>	timeout signal in SMBus mode
<i>I2C_FLAG_SMBALT</i>	SMBus alert status
<i>I2C_FLAG_MASTER</i>	a flag indicating whether I2C block is in master or slave mode
<i>I2C_FLAG_I2CBSY</i>	busy flag
<i>I2C_FLAG_TRS</i>	whether the I2C is a transmitter or a receiver
<i>I2C_FLAG_RXGC</i>	general call address (00h) received
<i>I2C_FLAG_DEFSMB</i>	default address of SMBus device
<i>I2C_FLAG_HSTSMB</i>	SMBus host header detected in slave mode
<i>I2C_FLAG_DUMOD</i>	dual flag in slave mode indicating which address is matched in dual-address mode
<i>I2C_FLAG_TFF</i>	txframe fall flag

<i>I2C_FLAG_TFR</i>	txframe rise flag
<i>I2C_FLAG_RFF</i>	rxframe fall flag
<i>I2C_FLAG_RFR</i>	rxframe rise flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```

/* check whether start condition send out */

FlagStatus flag_state = RESET;

flag_state = i2c_flag_get (I2C0, I2C_FLAG_SBSEND);

```

i2c_flag_clear

The description of i2c_flag_clear is shown as below:

Table 3-487. Function i2c_flag_clear

Function name	i2c_flag_clear
Function prototype	void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);
Function descriptions	clear I2C flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
flag	flag type
<i>I2C_FLAG_SMBALT</i>	SMBus Alert status
<i>I2C_FLAG_SMBTO</i>	timeout signal in SMBus mode
<i>I2C_FLAG_PECERR</i>	PEC error when receiving data
<i>I2C_FLAG_OUERR</i>	over-run or under-run situation occurs in slave mode

<i>I2C_FLAG_AERR</i>	acknowledge error
<i>I2C_FLAG_LOSTARB</i>	arbitration lost in master mode
<i>I2C_FLAG_BERR</i>	a bus error
<i>I2C_FLAG_ADDSEND</i>	cleared by reading I2C_STAT0 and reading I2C_STAT1
<i>I2C_FLAG_TFF</i>	txframe fall flag
<i>I2C_FLAG_TFR</i>	txframe rise flag
<i>I2C_FLAG_RFF</i>	rxframe fall flag
<i>I2C_FLAG_RFR</i>	rxframe rise flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear a bus error flag*/
i2c_flag_clear (I2C0, I2C_FLAG_BERR);
```

i2c_interrupt_enable

The description of i2c_interrupt_enable is shown as below:

Table 3-488. Function i2c_interrupt_enable

Function name	i2c_interrupt_enable
Function prototype	void i2c_interrupt_enable(uint32_t i2c_periph, uint32_t interrupt);
Function descriptions	enable I2C interrupt
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
interrupt	interrupt type

<i>I2C_INT_ERR</i>	error interrupt enable
<i>I2C_INT_EV</i>	event interrupt enable
<i>I2C_INT_BUF</i>	<i>buffer interrupt enable</i>
<i>I2C_INT_TFF</i>	<i>txframe fall interrupt enable</i>
<i>I2C_INT_TFR</i>	<i>txframe rise interrupt enable</i>
<i>I2C_INT_RFF</i>	<i>rxframe fall interrupt enable</i>
<i>I2C_INT_RFR</i>	<i>rxframe rise interrupt enable</i>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 error interrupt */
i2c_interrupt_enable (I2C0, I2C_INT_EV);
```

i2c_interrupt_disable

The description of i2c_interrupt_disable is shown as below:

Table 3-489. Function i2c_interrupt_disable

Function name	i2c_interrupt_disable
Function prototype	void i2c_interrupt_disable(uint32_t i2c_periph, uint32_t interrupt);
Function descriptions	disable I2C interrupt
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
interrupt	interrupt type
<i>I2C_INT_ERR</i>	error interrupt disable

<i>I2C_INT_EV</i>	event interrupt disable
<i>I2C_INT_BUF</i>	buffer interrupt disable
<i>I2C_INT_TFF</i>	<i>txframe fall interrupt enable</i>
<i>I2C_INT_TFR</i>	<i>txframe rise interrupt enable</i>
<i>I2C_INT_RFF</i>	<i>rxframe fall interrupt enable</i>
<i>I2C_INT_RFR</i>	<i>rxframe rise interrupt enable</i>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 error interrupt */
i2c_interrupt_disable (I2C0, I2C_INT_EV);
```

i2c_interrupt_flag_get

The description of i2c_interrupt_flag_get is shown as below:

Table 3-490. Function i2c_interrupt_flag_get

Function name	i2c_interrupt_flag_get
Function prototype	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, uint32_t int_flag);
Function descriptions	get I2C interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
<i>int_flag</i>	interrupt flag
<i>I2C_INT_FLAG_SBSEN</i>	start condition sent out in master mode interrupt flag

<i>I2C_INT_FLAG_ADDS END</i>	address is sent in master mode or received and matches in slave mode interrupt flag
<i>I2C_INT_FLAG_BTC</i>	byte transmission finishes
<i>I2C_INT_FLAG_ADD10 SEND</i>	header of 10-bit address is sent in master mode interrupt flag
<i>I2C_INT_FLAG_STPD ET</i>	stop condition detected in slave mode interrupt flag
<i>I2C_INT_FLAG_RBNE</i>	I2C_DATA is not Empty during receiving interrupt flag
<i>I2C_INT_FLAG_TBE</i>	I2C_DATA is empty during transmitting interrupt flag
<i>I2C_INT_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
<i>I2C_INT_FLAG_LOSTA RB</i>	arbitration lost in master mode interrupt flag
<i>I2C_INT_FLAG_AERR</i>	acknowledge error interrupt flag
<i>I2C_INT_FLAG_OUER R</i>	over-run or under-run situation occurs in slave mode interrupt flag
<i>I2C_INT_FLAG_PCE RR</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBT O</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBA LT</i>	SMBus Alert status interrupt flag
<i>I2C_INT_FLAG_TFF</i>	txframe fall interrupt flag
<i>I2C_INT_FLAG_TFR</i>	txframe rise interrupt flag
<i>I2C_INT_FLAG_RFF</i>	rxframe fall interrupt flag
<i>I2C_INT_FLAG_RFR</i>	rxframe rise interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

/* check the byte transmission finishes interrupt flag is set or not*/

```

FlagStatus flag_state = RESET;
flag_state = i2c_interrupt_flag_get (I2C0, I2C_INT_FLAG_BTC);
```

i2c_interrupt_flag_clear

The description of i2c_interrupt_flag_clear is shown as below:

Table 3-491. Function i2c_interrupt_flag_clear

Function name	i2c_interrupt_flag_clear
Function prototype	void i2c_interrupt_flag_clear(uint32_t i2c_periph, uint32_t int_flag);
Function descriptions	clear I2C interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
intflag	interrupt flag
I2C_INT_FLAG_ADDS END	address is sent in master mode or received and matches in slave mode interrupt flag
I2C_INT_FLAG_BERR	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
I2C_INT_FLAG_LOSTA RB	arbitration lost in master mode interrupt flag
I2C_INT_FLAG_AERR	acknowledge error interrupt flag
I2C_INT_FLAG_OUER R	over-run or under-run situation occurs in slave mode interrupt flag
I2C_INT_FLAG_PECCE RR	PEC error when receiving data interrupt flag
I2C_INT_FLAG_SMBTO	timeout signal in SMBus mode interrupt flag
I2C_INT_FLAG_SMBA LT	SMBus Alert status interrupt flag

<i>I2C_INT_FLAG_TFF</i>	txframe fall interrupt flag
<i>I2C_INT_FLAG_TFR</i>	txframe rise interrupt flag
<i>I2C_INT_FLAG_RFF</i>	rxframe fall interrupt flag
<i>I2C_INT_FLAG_RFR</i>	rxframe rise interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the acknowledge error interrupt flag */
i2c_interrupt_flag_clear (I2C0, I2C_INT_FLAG_AERR);
```

3.17. IPA

The IPA provides a configurable and flexible image format conversion from one or two source image to the destination image. The IPA registers are listed in chapter [3.17.1](#), the IPA firmware functions are introduced in chapter [3.17.2](#).

3.17.1. Descriptions of Peripheral registers

IPA registers are listed in the table shown as below:

Table 3-492. IPA Registers

Registers	Descriptions
IPA_CTL	IPA control register
IPA_INTF	IPA interrupt flag register
IPA_INTC	IPA interrupt flag clear register
IPA_FMADDR	IPA foreground memory base address register
IPA_FLOFF	IPA foreground line offset register
IPA_BMADDR	IPA background memory base address register
IPA_BLOFF	IPA background line offset register
IPA_FPCTL	IPA foreground pixel control register

Registers	Descriptions
IPA_FPV	IPA foreground pixel value register
IPA_BPCTL	IPA background pixel control register
IPA_BPV	IPA background pixel value register
IPA_FLMADDR	IPA foreground LUT memory base address register
IPA_BLMADDR	IPA background LUT memory base address register
IPA_DPCTL	IPA destination pixel control register
IPA_DPV	IPA destination pixel value register
IPA_DMADDR	IPA destination memory base address register
IPA_DLOFF	IPA destination line offset register
IPA_IMS	IPA image size register
IPA_LM	IPA line mark register
IPA_ITCTL	IPA inter-timer control register

3.17.2. Descriptions of Peripheral functions

IPA firmware functions are listed in the table shown as below:

Table 3-493. IPA firmware function

Function name	Function description
ipa_deinit	deinitialize IPA
ipa_transfer_enable	enable IPA transfer
ipa_transfer_hangup_enable	enable IPA transfer hang up
ipa_transfer_hangup_disable	disable IPA transfer hang up
ipa_transfer_stop_enable	enable IPA transfer stop
ipa_transfer_stop_disable	disable IPA transfer stop
ipa_foreground_lut_loading_enable	enable IPA foreground LUT loading
ipa_background_lut_loading_enable	enable IPA background LUT loading
ipa_pixel_format_convert_mode_set	set pixel format convert mode, the function is invalid when the IPA transfer is enabled
ipa_foreground_struct_para_init	initialize the structure of IPA foreground parameter struct with

Function name	Function description
	the default values, it is suggested that call this function after an ipa_foreground_parameter_struct structure is defined
ipa_foreground_init	initialize foreground parameters
ipa_background_struct_para_init	initialize the structure of IPA background parameter struct with the default values, it is suggested that call this function after an ipa_background_parameter_struct structure is defined
ipa_background_init	initialize background parameters
ipa_destination_struct_para_init	initialize the structure of IPA destination parameter struct with the default values, it is suggested that call this function after an ipa_destination_parameter_struct structure is defined
ipa_destination_init	initialize destination parameters
ipa_foreground_lut_init	initialize IPA foreground LUT parameters
ipa_background_lut_init	initialize IPA background LUT parameters
ipa_line_mark_config	configure IPA line mark
ipa_inter_timer_config	IPA inter-timer enable or disable
ipa_interval_clock_num_config	configure the number of clock cycles interval
ipa_flag_get	get IPA flag status in IPA_INTF register
ipa_flag_clear	clear IPA flag in IPA_INTF register
ipa_interrupt_enable	enable IPA interrupt
ipa_interrupt_disable	disable IPA interrupt
ipa_interrupt_flag_get	get IPA interrupt flag
ipa_interrupt_flag_clear	clear IPA interrupt flag

Structure ipa_foreground_parameter_struct

Table3-494. Structure ipa_foreground_parameter_struct

Member name	Function description
foreground_memaddr	foreground memory base address
foreground_lineoff	foreground line offset
foreground_prealpha	foreground pre-defined alpha value

Member name	Function description
foreground_alpha_algorithm	foreground alpha value calculation algorithm
foreground_pf	foreground pixel format
foreground_prered	foreground pre-defined red value
foreground_pregreen	foreground pre-defined green value
foreground_preblue	foreground pre-defined blue value

Structure ipa_background_parameter_struct

Table3-495. Structure ipa_background_parameter_struct

Member name	Function description
background_memaddr	background memory base address
background_lineoff	background line offset
background_prealpha	background pre-defined alpha value
background_alpha_algorithm	background alpha value calculation algorithm
background_pf	background pixel format
background_prered	background pre-defined red value
background_pregreen	background pre-defined green value
background_preblue	background pre-defined blue value

Structure ipa_destination_parameter_struct

Table3-496. Structure ipa_destination_parameter_struct

Member name	Function description
destination_memaddr	destination memory base address
destination_lineoff	destination line offset
destination_prealpha	destination pre-defined alpha value
destination_pf	destination pixel format
destination_prered	destination pre-defined red value
destination_pregreen	destination pre-defined green value
destination_preblue	destination pre-defined blue value

Member name	Function description
image_width	width of the image to be processed
image_height	height of the image to be processed

ipa_deinit

The description of ipa_deinit is shown as below:

Table 3-497. Function ipa_deinit

Function name	ipa_deinit
Function prototype	void ipa_deinit(void);
Function descriptions	deinitialize IPA registers
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize IPA */

ipa_deinit();
```

ipa_transfer_enable

The description of ipa_transfer_enable is shown as below:

Table 3-498. Function ipa_transfer_enable

Function name	ipa_transfer_enable
Function prototype	void ipa_transfer_enable(void);
Function descriptions	enable IPA transfer
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IPA transfer */
ipa_transfer_enable();
```

ipa_transfer_hangup_enable

The description of ipa_transfer_hangup_enable is shown as below:

Table 3-499. Function ipa_transfer_hangup_enable

Function name	ipa_transfer_hangup_enable
Function prototype	void ipa_transfer_hangup_enable(void);
Function descriptions	enable IPA transfer hang up
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IPA transfer hang up */
ipa_transfer_hangup_enable();
```

ipa_transfer_hangup_disable

The description of ipa_transfer_hangup_disable is shown as below:

Table 3-500. Function ipa_transfer_hangup_disable

Function name	ipa_transfer_hangup_disable
Function prototype	void ipa_transfer_hangup_disable(void);
Function descriptions	disable IPA transfer hang up
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable IPA transfer hang up */
ipa_transfer_hangup_disable();
```

ipa_transfer_stop_enable

The description of ipa_transfer_stop_enable is shown as below:

Table 3-501. Function ipa_transfer_stop_enable

Function name	ipa_transfer_stop_enable
Function prototype	void ipa_transfer_stop_enable(void);
Function descriptions	enable IPA transfer stop
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable IPA transfer stop */
ipa_transfer_stop_enable();
```

ipa_transfer_stop_disable

The description of ipa_transfer_stop_disable is shown as below:

Table 3-502. Function ipa_transfer_stop_disable

Function name	ipa_transfer_stop_disable
Function prototype	void ipa_transfer_stop_disable(void);
Function descriptions	disable IPA transfer stop
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable IPA transfer stop */
ipa_transfer_stop_disable();
```

ipa_foreground_lut_loading_enable

The description of ipa_foreground_lut_loading_enable is shown as below:

Table 3-503. Function ipa_foreground_lut_loading_enable

Function name	ipa_foreground_lut_loading_enable
Function prototype	void ipa_foreground_lut_loading_enable(void);

Function descriptions	enable IPA foreground LUT loading
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IPA foreground LUT loading */
ipa_foreground_lut_loading_enable();
```

ipa_background_lut_loading_enable

The description of ipa_background_lut_loading_enable is shown as below:

Table 3-504. Function ipa_background_lut_loading_enable

Function name	ipa_background_lut_loading_enable
Function prototype	void ipa_background_lut_loading_enable(void);
Function descriptions	enable IPA background LUT loading
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IPA background LUT loading */
```

```
ipa_background_lut_loading_enable();
```

ipa_pixel_format_convert_mode_set

The description of ipa_pixel_format_convert_mode_set is shown as below:

Table 3-505. Function ipa_pixel_format_convert_mode_set

Function name	ipa_pixel_format_convert_mode_set
Function prototype	void ipa_pixel_format_convert_mode_set(uint32_t pfcm);
Function descriptions	set pixel format convert mode, the function is invalid when the IPA transfer is enabled
Precondition	-
The called functions	-
Input parameter{in}	
pfcm	pixel format convert mode
<i>IPA_FGTODE</i>	foreground memory to destination memory without pixel format convert
<i>IPA_FGTODE_PF_CO_NVERT</i>	foreground memory to destination memory with pixel format convert
<i>IPA_FGBGTODE</i>	foreground and background memory to destination memory
<i>IPA_FILL_UP_DE</i>	fill up destination memory with specific color
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure foreground memory to destination memory with pixel format convert */
ipa_pixel_format_convert_mode_set(IPA_FGTODE_PF_CONVERT);
```

ipa_foreground_struct_para_init

The description of ipa_foreground_struct_para_init is shown as below:

Table 3-506. Function ipa_foreground_struct_para_init

Function name	ipa_foreground_struct_para_init
Function prototype	void ipa_foreground_struct_para_init(ipa_foreground_parameter_struct*

	foreground_struct);
Function descriptions	initialize the structure of IPA foreground parameter struct with the default values, it is suggested that call this function after an ipa_foreground_parameter_struct structure is defined
Precondition	-
The called functions	-
Input parameter{in}	
*foreground_struct	a pointer to ipa_foreground_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
ipa_foreground_parameter_struct fg_struct;
/* initialize the structure of IPA foreground parameter struct with the default values */
ipa_foreground_struct_para_init(&fg_struct);
```

ipa_foreground_init

The description of ipa_foreground_init is shown as below:

Table 3-507. Function ipa_foreground_init

Function name	ipa_foreground_init
Function prototype	void ipa_foreground_init(ipa_foreground_parameter_struct* foreground_struct);
Function descriptions	initialize foreground parameters
Precondition	-
The called functions	-
Input parameter{in}	
*foreground_struct	a pointer to ipa_foreground_parameter_struct
Output parameter{out}	
-	-

Return value	
-	-

Example:

```

ipa_foreground_parameter_struct    ipa_fg_init_struct;
ipa_foreground_struct_para_init(&ipa_fg_init_struct);

/* configure IPA foreground */

ipa_fg_init_struct.foreground_memaddr = (uint32_t)&gBuffer;
ipa_fg_init_struct.foreground_pf = FOREGROUND_PPF_RGB565;
ipa_fg_init_struct.foreground_alpha_algorithm = IPA_FG_ALPHA_MODE_1;
ipa_fg_init_struct.foreground_prealpha = 0x75;
ipa_fg_init_struct.foreground_lineoff = 0x00;
ipa_fg_init_struct.foreground_preblue = 0x00;
ipa_fg_init_struct.foreground_pregreen = 0x00;
ipa_fg_init_struct.foreground_prered = 0x00;

/* foreground initialization */

ipa_foreground_init(&ipa_fg_init_struct);

```

ipa_background_struct_para_init

The description of ipa_background_struct_para_init is shown as below:

Table 3-508. Function ipa_background_struct_para_init

Function name	ipa_background_struct_para_init
Function prototype	void ipa_background_struct_para_init(ipa_background_parameter_struct* background_struct);
Function descriptions	initialize the structure of IPA background parameter struct with the default values , it is suggested that call this function after an ipa_background_parameter_struct structure is defined
Precondition	-
The called functions	-
Input parameter{in}	
*background_struct	a pointer to ipa_background_parameter_struct

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
ipa_background_parameter_struct bg_struct;
/* initialize the structure of IPA background parameter struct with the default values */
ipa_background_struct_para_init(&bg_struct);
```

ipa_background_init

The description of ipa_background_init is shown as below:

Table 3-509. Function ipa_background_init

Function name	ipa_background_init
Function prototype	void ipa_background_init(ipa_background_parameter_struct* background_struct);
Function descriptions	initialize background parameters
Precondition	-
The called functions	-
Input parameter{in}	
*background_struct	a pointer to ipa_background_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
ipa_background_parameter_struct ipa_bg_init_struct;
ipa_background_struct_para_init(&ipa_bg_init_struct);
/* configure IPA background */
ipa_bg_init_struct.background_memaddr = (uint32_t)&gBuffer;
```

```

ipa_bg_init_struct.background_pf = BACKGROUND_PPF_RGB565;
ipa_bg_init_struct.background_alpha_algorithm = IPA_BG_ALPHA_MODE_0;
ipa_bg_init_struct.background_prealpha = 255;
ipa_bg_init_struct.background_lineoff = 0x00;
ipa_bg_init_struct.background_preblue = 0x00;
ipa_bg_init_struct.background_pregreen = 0x00;
ipa_bg_init_struct.background_prered = 0x00;
/* background initialization */
ipa_background_init(&ipa_bg_init_struct);

```

ipa_destination_struct_para_init

The description of ipa_destination_struct_para_init is shown as below:

Table 3-510. Function ipa_destination_struct_para_init

Function name	ipa_destination_struct_para_init
Function prototype	void ipa_destination_struct_para_init(ipa_destination_parameter_struct* destination_struct);
Function descriptions	initialize the structure of IPA destination parameter struct with the default values, it is suggested that call this function after an ipa_destination_parameter_struct structure is defined
Precondition	-
The called functions	-
Input parameter{in}	
*destination_struct	a pointer to ipa_destination_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

ipa_destination_parameter_struct destination_struct;
/* initialize the structure of IPA destination parameter struct with the default values */

```

```
ipa_destination_struct_para_init(&destination_struct);
```

ipa_destination_init

The description of ipa_destination_init is shown as below:

Table 3-511. Function ipa_destination_init

Function name	ipa_destination_init
Function prototype	void ipa_destination_init(ipa_destination_parameter_struct* destination_struct);
Function descriptions	initialize destination parameters
Precondition	-
The called functions	-
Input parameter{in}	
*destination_struct	a pointer to ipa_destination_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
ipa_destination_parameter_struct ipa_destination_init_struct;
ipa_destination_struct_para_init(&ipa_destination_init_struct);

/* configure destination pixel format */
ipa_destination_init_struct.destination_pf = IPA_DPF_RGB565;

/* configure destination memory base address */
ipa_destination_init_struct.destination_memaddr = (uint32_t)&gBuffer;

/* configure destination pre-defined alpha value RGB */
ipa_destination_init_struct.destination_pregreen = 0;

ipa_destination_init_struct.destination_preblue = 0;

ipa_destination_init_struct.destination_prered = 0;

ipa_destination_init_struct.destination_prealpha = 0;

/* configure destination line offset */
```

```

ipa_destination_init_struct.destination_lineoff = 0;

/* configure height of the image to be processed */

ipa_destination_init_struct.image_height = 160;

/* configure width of the image to be processed */

ipa_destination_init_struct.image_width = 229;

/* ipa destination initialization */

ipa_destination_init(&ipa_destination_init_struct);

```

ipa_foreground_lut_init

The description of ipa_foreground_lut_init is shown as below:

Table 3-512. Function ipa_foreground_lut_init

Function name	ipa_foreground_lut_init
Function prototype	void ipa_foreground_lut_init(uint8_t fg_lut_num,uint8_t fg_lut_pf, uint32_t fg_lut_addr);
Function descriptions	initialize IPA foreground LUT parameters
Precondition	-
The called functions	-
Input parameter{in}	
fg_lut_num	foreground LUT number of pixel
Input parameter{in}	
fg_lut_pf	foreground LUT pixel format
<i>IPA_LUT_PF_ARGB88 88</i>	LUT pixel format is ARGB8888
<i>IPA_LUT_PF_RGB888</i>	LUT pixel format is RGB888
Input parameter{in}	
fg_lut_addr	foreground LUT memory base address. The address must be aligned to 8-bit, 16-bit or 32-bit corresponding with the foreground LUT pixel format.
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* initialize the LUT foreground LUT parameters */

ipa_foreground_lut_init(1, IPA_LUT_PF_ARGB8888, 0x20002000);
```

ipa_background_lut_init

The description of ipa_background_lut_init is shown as below:

Table 3-513. Function ipa_background_lut_init

Function name	ipa_background_lut_init
Function prototype	void ipa_background_lut_init(uint8_t bg_lut_num,uint8_t bg_lut_pf, uint32_t bg_lut_addr);
Function descriptions	initialize IPA background LUT parameters
Precondition	-
The called functions	-
Input parameter{in}	
bg_lut_num	background LUT number of pixel
Input parameter{in}	
bg_lut_pf	background LUT pixel format
<i>IPA_LUT_PF_ARGB8888</i>	LUT pixel format is ARGB8888
<i>IPA_LUT_PF_RGB888</i>	LUT pixel format is RGB888
Input parameter{in}	
bg_lut_addr	background LUT memory base address. The address must be aligned to 8-bit, 16-bit or 32-bit corresponding with the foreground LUT pixel format.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the LUT background LUT parameters */
```

```
ipa_background_lut_init(2, IPA_LUT_PF_RGB888, 0x20001000);
```

ipa_line_mark_config

The description of ipa_line_mark_config is shown as below:

Table 3-514. Function ipa_line_mark_config

Function name	ipa_line_mark_config
Function prototype	void ipa_line_mark_config(uint16_t line_num);
Function descriptions	configure IPA line mark
Precondition	-
The called functions	-
Input parameter{in}	
line_num	line number
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure line mark */
ipa_line_mark_config(10);
```

ipa_inter_timer_config

The description of ipa_inter_timer_config is shown as below:

Table 3-515. Function ipa_inter_timer_config

Function name	ipa_inter_timer_config
Function prototype	void ipa_inter_timer_config(uint8_t timer_cfg);
Function descriptions	inter-timer enable or disable
Precondition	-
The called functions	-
Input parameter{in}	
timer_cfg	inter-timer configuration

<i>IPA_INTER_TIMER_ENABLE</i>	enable inter-timer
<i>IPA_INTER_TIMER_DISABLE</i>	disable inter-timer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable inter-timer */

ipa_inter_timer_config(IPA_INTER_TIMER_ENABLE);
```

ipa_interval_clock_num_config

The description of ipa_interval_clock_num_config is shown as below:

Table 3-516. Function ipa_interval_clock_num_config

Function name	ipa_interval_clock_num_config
Function prototype	void ipa_interval_clock_num_config(uint8_t clk_num);
Function descriptions	configure the number of clock cycles interval
Precondition	-
The called functions	-
Input parameter{in}	
clk_num	the number of clock cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the number of clock cycles interval and it has no meaning if ITEN is '0' */

ipa_interval_clock_num_config(1);
```

ipa_flag_get

The description of ipa_flag_get is shown as below:

Table 3-517. Function ipa_flag_get

Function name	ipa_flag_get
Function prototype	FlagStatus ipa_flag_get(uint32_t flag);
Function descriptions	get IPA flag status in IPA_INTF register
Precondition	-
The called functions	-
Input parameter{in}	
flag	IPA flags
<i>IPA_FLAG_TAE</i>	transfer access error interrupt flag
<i>IPA_FLAG_FTF</i>	full transfer finish interrupt flag
<i>IPA_FLAG_TLM</i>	transfer line mark interrupt flag
<i>IPA_FLAG_LAC</i>	LUT access conflict interrupt flag
<i>IPA_FLAG_LLFI</i>	LUT loading finish interrupt flag
<i>IPA_FLAG_WCF</i>	wrong configuration interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* wait for full transfer finish flag set */

while(ipa_flag_get(IPA_FLAG_FTF) == RESET);
```

ipa_flag_clear

The description of ipa_flag_clear is shown as below:

Table 3-518. Function ipa_flag_clear

Function name	ipa_flag_clear
Function prototype	void ipa_flag_clear(uint32_t flag);

Function descriptions	clear IPA flag in IPA_INTF register
Precondition	-
The called functions	-
Input parameter{in}	
flag	IPA flags
<i>IPA_FLAG_TAE</i>	transfer access error interrupt flag
<i>IPA_FLAG_FTF</i>	full transfer finish interrupt flag
<i>IPA_FLAG_TLM</i>	transfer line mark interrupt flag
<i>IPA_FLAG_LAC</i>	LUT access conflict interrupt flag
<i>IPA_FLAG_LLFI</i>	LUT loading finish interrupt flag
<i>IPA_FLAG_WCF</i>	wrong configuration interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* wait for full transfer finish flag set */

while(ipa_flag_get(IPA_FLAG_FTF) == RESET);

/* clear full transfer finish flag */

ipa_flag_clear(IPA_FLAG_FTF);

```

ipa_interrupt_enable

The description of ipa_interrupt_enable is shown as below:

Table 3-519. Function ipa_interrupt_enable

Function name	ipa_interrupt_enable
Function prototype	void ipa_interrupt_enable(uint32_t int_flag);
Function descriptions	enable IPA interrupt
Precondition	-
The called functions	-

Input parameter{in}	
int_flag	IPA interrupt flags
<i>IPA_INT_TAE</i>	transfer access error interrupt
<i>IPA_INT_FTF</i>	full transfer finish interrupt
<i>IPA_INT_TLM</i>	transfer line mark interrupt
<i>IPA_INT_LAC</i>	LUT access conflict interrupt
<i>IPA_INT_LLFI</i>	LUT loading finish interrupt
<i>IPA_INT_WCF</i>	wrong configuration interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable full transfer finish interrupt */
ipa_interrupt_enable(IPA_INT_FTF);
```

ipa_interrupt_disable

The description of ipa_interrupt_disable is shown as below:

Table 3-520. Function ipa_interrupt_disable

Function name	ipa_interrupt_disable
Function prototype	void ipa_interrupt_disable(uint32_t int_flag);
Function descriptions	disable IPA interrupt
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	IPA interrupt flags
<i>IPA_INT_TAE</i>	transfer access error interrupt
<i>IPA_INT_FTF</i>	full transfer finish interrupt
<i>IPA_INT_TLM</i>	transfer line mark interrupt

<i>IPA_INT_LAC</i>	LUT access conflict interrupt
<i>IPA_INT_LLFI</i>	LUT loading finish interrupt
<i>IPA_INT_WCF</i>	wrong configuration interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable full transfer finish interrupt */

ipa_interrupt_disable(IPA_INT_FTF);
```

ipa_interrupt_flag_get

The description of ipa_interrupt_flag_get is shown as below:

Table 3-521. Function ipa_interrupt_flag_get

Function name	ipa_interrupt_flag_get
Function prototype	FlagStatus ipa_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get IPA interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
<i>int_flag</i>	IPA interrupt flag flags
<i>IPA_INT_FLAG_TAE</i>	transfer access error interrupt flag
<i>IPA_INT_FLAG_FTF</i>	full transfer finish interrupt flag
<i>IPA_INT_FLAG_TLM</i>	transfer line mark interrupt flag
<i>IPA_INT_FLAG_LAC</i>	LUT access conflict interrupt flag
<i>IPA_INT_FLAG_LLFI</i>	LUT loading finish interrupt flag
<i>IPA_INT_FLAG_WCF</i>	wrong configuration interrupt flag
Output parameter{out}	
-	-

Return value	
FlagStatus	SET or RESET

Example:

```
/* check whether full transfer finish interrupt flag is SET */

while(ipa_interrupt_flag_get(IPA_INT_FLAG_FTF) == RESET);
```

ipa_interrupt_flag_clear

The description of ipa_interrupt_flag_clear is shown as below:

Table 3-522. Function ipa_interrupt_flag_clear

Function name	ipa_interrupt_flag_clear
Function prototype	void ipa_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear IPA interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
<i>int_flag</i>	IPA interrupt flag flags
<i>IPA_INT_FLAG_TAE</i>	transfer access error interrupt flag
<i>IPA_INT_FLAG_FTF</i>	full transfer finish interrupt flag
<i>IPA_INT_FLAG_TLM</i>	transfer line mark interrupt flag
<i>IPA_INT_FLAG_LAC</i>	LUT access conflict interrupt flag
<i>IPA_INT_FLAG_LLFI</i>	LUT loading finish interrupt flag
<i>IPA_INT_FLAG_WCF</i>	wrong configuration interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
if(ipa_interrupt_flag_get(IPA_INT_FLAG_FTF) != RESET){
```

```
    /* clear full transfer finish interrupt flag */
```

```

ipa_interrupt_flag_clear(IPA_INT_FLAG_FTF);
}

```

3.18. IREF

IREF is a software package that provide the interfaces for IREF. The IREF registers are listed in chapter [3.18.1](#), the IREF firmware functions are introduced in chapter [3.18.2](#).

3.18.1. Descriptions of Peripheral registers

Table 3-523. IREF Registers

Registers	Descriptions
IREF_CTL	Control register

3.18.2. Descriptions of Peripheral functions

IREF firmware functions are listed in the table shown as below:

Table 3-524. IREF firmware function

Function name	Function description
iref_deinit	deinit IREF
iref_enable	enable IREF
iref_disable	disable IREF
iref_mode_set	set IREF mode
iref_sink_set	set IREF sink mode
iref_precision_trim_value_set	set IREF precision_trim_value
iref_step_data_config	set IREF step data

iref_deinit

The description of iref_deinit is shown as below:

Table 3-525. Function iref_deinit

Function name	iref_deinit
Function prototype	void iref_deinit (void);
Function descriptions	Reset IREF
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* reset IREF */

iref_deinit();
```

iref_enable

The description of iref_enable is shown as below:

Table 3-526. Function iref_enable

Function name	iref_enable
Function prototype	void iref_enable (void);
Function descriptions	Enable IREF
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IREF */

iref_enable();
```

iref_disable

The description of iref_disable is shown as below:

Table 3-527. Function iref_disable

Function name	iref_disable
Function prototype	void iref_disable(void);
Function descriptions	Disable IREF
Precondition	-
The called functions	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable IREF */
```

```
iref_disable();
```

iref_mode_set

The description of iref_mode_set is shown as below:

Table 3-528. Function iref_mode_set

Function name	iref_mode_set
Function prototype	void iref_mode_set(uint32_t step);
Function descriptions	set IREF mode
Precondition	-
The called functions	-
Input parameter{in}	
step	IREF mode
<i>IREF_MODE_LOW_POWER</i>	Low power mode,1uA step
<i>IREF_MODE_HIGH_CURRENT</i>	High current mode,8uA step
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set IREF mode */
```

```
iref_mode_set(IREF_MODE_LOW_POWER);
```

iref_sink_set

The description of iref_sink_set is shown as below:

Table 3-529. Function iref_sink_set

Function name	iref_sink_set
Function prototype	void iref_sink_set(uint32_t sinkmode);
Function descriptions	set IREF sink mode
Precondition	-
The called functions	-
Input parameter{in}	
sinkmode	Sink mode
<i>IREF_SOURCE_CU RRENT</i>	source current
<i>IREF_SINK_CURR ENT</i>	sink current
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set IREF source current mode */
iref_sink_set(IREF_SOURCE_CURRENT);
```

iref_precision_trim_value_set

The description of iref_precision_trim_value_set is shown as below:

Table 3-530. Function iref_precision_trim_value_set

Function name	iref_precision_trim_value_set
Function prototype	void iref_precision_trim_value_set (uint32_t precisiontrim);
Function descriptions	set IREF precision trim value
Precondition	-
The called functions	-
Input parameter{in}	
precisiontrim	IREF precision_trim_value
<i>IREF_CUR_PRECI SION_TRIM_X</i>	(X=0..31): (-15.. +16)%
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set IREF precision trim value -15% */

iref_precision_trim_value_set(IREF_CUR_PRECISION_TRIM_0);
```

iref_step_data_config

The description of iref_step_data_config is shown as below:

Table 3-531. Function iref_step_data_config

Function name	iref_step_data_config
Function prototype	void iref_step_data_config (uint32_t stepdata);
Function descriptions	set IREF step data
Precondition	-
The called functions	-
Input parameter{in}	
stepdata	Step data
IREF_CUR_STEP_DATA_X	(X=0..63): step*x
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set IREF step data */

iref_step_data_config(IREF_CUR_STEP_DATA_0);
```

3.19. MISC

MISC is a software package that provide the interfaces for NVIC and SysTick. The NVIC and SysTick registers are listed in chapter [3.19.1](#), the MISC firmware functions are introduced in chapter [3.19.2](#).

3.19.1. Descriptions of Peripheral registers

Table 3-532. NVIC Registers

Registers	Descriptions
ISER ⁽¹⁾	Interrupt Set Enable Register
ICER ⁽¹⁾	Interrupt Clear Enable Register
ISPR ⁽¹⁾	Interrupt Set Pending Register

Registers	Descriptions
ICPR ⁽¹⁾	Interrupt Clear Pending Register
IABR ⁽¹⁾	Interrupt Active bit Register
IP ⁽¹⁾	Interrupt Priority Register
STIR ⁽¹⁾	Software Trigger Interrupt Register
CPUID ⁽²⁾	CPUID Base Register
ICSR ⁽²⁾	Interrupt Control and State Register
VTOR ⁽²⁾	Vector Table Offset Register
AIRCR ⁽²⁾	Application Interrupt and Reset Control Register
SCR ⁽²⁾	System Control Register
CCR ⁽²⁾	Configuration Control Register
SHP ⁽²⁾	System Handlers Priority Registers
SHCSR ⁽²⁾	System Handler Control and State Register
CFSR ⁽²⁾	Configurable Fault Status Register
HFSR ⁽²⁾	HardFault Status Register
DFSR ⁽²⁾	Debug Fault Status Register
MMFAR ⁽²⁾	MemManage Fault Address Register
BFAR ⁽²⁾	BusFault Address Register
AFSR ⁽²⁾	Auxiliary Fault Status Register
PFR ⁽²⁾	Processor Feature Register
DFR ⁽²⁾	Debug Feature Register
ADR ⁽²⁾	Auxiliary Feature Register
MMFR ⁽²⁾	Memory Model Feature Register
ISAR ⁽²⁾	Instruction Set Attributes Register
CPACR ⁽²⁾	Coprocessor Access Control Register

1. refer to the structure NVIC_Type, is defined in the core_cm4.h file

2. refer to the structure SCB_Type, is defined in the core_cm4.h file

Table 3-533. SysTick Registers

Registers	Descriptions
CTRL ⁽¹⁾	SysTick Control and Status Register
LOAD ⁽¹⁾	SysTick Reload Value Register
VAL ⁽¹⁾	SysTick Current Value Register
CALIB ⁽¹⁾	SysTick Calibration Register

1. refer to the structure SysTick_Type, is defined in the core_cm4.h file

3.19.2. Descriptions of Peripheral functions

Enum IRQn_Type

Table 3-534. Enum IRQn_Type

Member name	Function description
WWDGT_IRQn	window watchDog timer interrupt

Member name	Function description
LVD IRQn	LVD through EXTI line detect interrupt
TAMPER_STAMP_IRQn	tamper through EXTI line detect
RTC_WKUP_IRQn	RTC through EXTI line interrupt
FMC_IRQn	FMC interrupt
RCU_CTC_IRQn	RCU and CTC interrupt
EXTI0_IRQn	EXTI line 0 interrupts
EXTI1_IRQn	EXTI line 1 interrupts
EXTI2_IRQn	EXTI line 2 interrupts
EXTI3_IRQn	EXTI line 3 interrupts
EXTI4_IRQn	EXTI line 4 interrupts
DMA0_Channel0_IRQn	DMA0 channel0 interrupt
DMA0_Channel1_IRQn	DMA0 channel1 interrupt
DMA0_Channel2_IRQn	DMA0 channel2 interrupt
DMA0_Channel3_IRQn	DMA0 channel3 interrupt
DMA0_Channel4_IRQn	DMA0 channel4 interrupt
DMA0_Channel5_IRQn	DMA0 channel5 interrupt
DMA0_Channel6_IRQn	DMA0 channel6 interrupt
ADC_IRQn	ADC interrupt
CAN0_TX_IRQn	CAN0 transmit interrupts
CAN0_RX0_IRQn	CAN0 receive0 interrupts
CAN0_RX1_IRQn	CAN0 receive1 interrupts
CAN0_EWMC_IRQn	CAN0 EWMC interrupts
EXTI5_9_IRQn	EXTI[9:5] interrupts
TIMER0_BRK_TIMER8_IRQn	TIMER0 break and TIMER8 interrupts
TIMER0_UP_TIMER9_IRQn	TIMER0 update and TIMER9 interrupts
TIMER0_TRG_CMT_TIMER10_IRQn	TIMER0 trigger and commutation and TIMER10 interrupts
TIMER0_Channel_IRQn	TIMER0 channel capture compare interrupt
TIMER1_IRQn	TIMER1 interrupt
TIMER2_IRQn	TIMER2 interrupt
TIMER3_IRQn	TIMER3 interrupt
I2C0_EV_IRQn	I2C0 event interrupt
I2C0_ER_IRQn	I2C0 error interrupt
I2C1_EV_IRQn	I2C1 event interrupt
I2C1_ER_IRQn	I2C1 error interrupt
SPI0_IRQn	SPI0 interrupt
SPI1_IRQn	SPI1 interrupt
USART0_IRQn	USART0 interrupt
USART1_IRQn	USART1 interrupt
USART2_IRQn	USART2 interrupt
EXTI10_15_IRQn	EXTI[15:10] interrupts
RTC_Alarm_IRQn	RTC alarm interrupt

Member name	Function description
USBFS_WKUP_IRQn	USBFS wakeup interrupt
TIMER7_BRK_TIMER11_IRQn	TIMER7 break and TIMER11 interrupts
TIMER7_UP_TIMER12_IRQn	TIMER7 update and TIMER12 interrupts
TIMER7_TRG_CMT_TIMER13_IRQn	TIMER7 trigger and commutation and TIMER13 interrupts
TIMER7_Channel_IRQn	TIMER7 channel capture compare interrupt
DMA0_Channel7_IRQn	DMA0 channel7 interrupt
EXMC_IRQn	EXMC global interrupt
SDIO_IRQn	SDIO global interrupt
TIMER4_IRQn	TIMER4 global interrupt
SPI2_IRQn	SPI2 global interrupt
UART3_IRQn	UART3 global interrupt
UART4_IRQn	UART4 global interrupt
TIMER5_DAC_IRQn	TIMER5 and DAC global interrupt
TIMER6_IRQn	TIMER6 global interrupt
DMA1_Channel0_IRQn	DMA1 channel0 global interrupt
DMA1_Channel1_IRQn	DMA1 channel1 global interrupt
DMA1_Channel2_IRQn	DMA1 channel2 global interrupt
DMA1_Channel3_IRQn	DMA1 channel3 global interrupt
DMA1_Channel4_IRQn	DMA1 channel4 global interrupt
ENET_IRQn	ENET interrupt
ENET_WKUP_IRQn	ENET wakeup interrupt
CAN1_TX_IRQn	CAN1 transmit interrupt
CAN1_RX0_IRQn	CAN1 receive0 interrupt
CAN1_RX1_IRQn	CAN1 receive1 interrupt
CAN1_EWMC_IRQn	CAN1 EWMC interrupt
USBFS_IRQn	USBFS interrupt
DMA1_Channel5_IRQn	DMA1 Channel5 interrupt
DMA1_Channel6_IRQn	DMA1 Channel6 interrupt
DMA1_Channel7_IRQn	DMA1 Channel7 interrupt
USART5_IRQn	USART5 interrupt
I2C2_EV_IRQn	I2C2 event interrupt
I2C2_ER_IRQn	I2C3 error interrupt
USBHS_EP1_Out_IRQn	USBHS endpoint 1 out interrupt
USBHS_EP1_In_IRQn	USBHS endpoint 1 in interrupt
USBHS_WKUP_IRQn	USBHS Wakeup interrupt
USBHS_IRQn	USBHS interrupt
DCI_IRQn	DCI interrupt
TRNG_IRQn	TRNG interrupt
FPU_IRQn	FPU interrupt
UART6_IRQn	UART6 interrupt
UART7_IRQn	UART7 interrupt

Member name	Function description
SPI3_IRQHandler	SPI3 interrupt
SPI4_IRQHandler	SPI4 interrupt
SPI5_IRQHandler	SPI5 interrupt
TLI_IRQHandler	TLI interrupt
TLI_Error_IRQHandler	TLI error interrupt
IPA_IRQHandler	IPA interrupt

MISC firmware functions are listed in the table shown as below:

Table 3-535. MISC firmware function

Function name	Function description
nvic_priority_group_set	set the priority group
nvic_irq_enable	enable NVIC interrupt request
nvic_irq_disable	disable NVIC interrupt request
nvic_vector_table_set	set the NVIC vector table address
system_lowpower_set	set the state of the low power mode
system_lowpower_reset	reset the state of the low power mode
systick_clocksource_set	set the systick clock source

nvic_priority_group_set

The description of nvic_priority_group_set is shown as below:

Table 3-536. Function nvic_priority_group_set

Function name	nvic_priority_group_set
Function prototype	void nvic_priority_group_set(uint32_t nvic_prigroup);
Function descriptions	configure bits length of the priority group
Precondition	-
The called functions	-
Input parameter{in}	
nvic_prigroup	priority group
NVIC_PRIGROUP_PR_E0_SUB4	0 bits for pre-emption priority 4 bits for subpriority
NVIC_PRIGROUP_PR_E1_SUB3	1 bits for pre-emption priority 3 bits for subpriority
NVIC_PRIGROUP_PR_E2_SUB2	2 bits for pre-emption priority 2 bits for subpriority
NVIC_PRIGROUP_PR_E3_SUB1	3 bits for pre-emption priority 1 bits for subpriority
NVIC_PRIGROUP_PR_E4_SUB0	4 bits for pre-emption priority 0 bits for subpriority
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* priority group configuration , 0 bits for pre-emption priority 4 bits for subpriority */
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

nvic_irq_enable

The description of nvc_irq_enable is shown as below:

Table 3-537. Function nvc_irq_enable

Function name	nvc_irq_enable
Function prototype	void nvc_irq_enable(uint8_t nvc_irq, uint8_t nvc_irq_pre_priority, uint8_t nvc_irq_sub_priority);
Function descriptions	enable NVIC request, configure the priority of interrupt
Precondition	-
The called functions	nvc_priority_group_set
Input parameter{in}	
nvc_irq	NVIC interrupt, refer to enum Enum IRQn_Type
Input parameter{in}	
nvc_irq_pre_priority	the pre-emption priority needed to set (0~4)
Input parameter{in}	
nvc_irq_sub_priority	the subpriority needed to set (0~4)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable window watchDog timer interrupt , pre-emption priority is 1, subpriority is 1 */
nvc_irq_enable(WWDGT_IRQn, 1, 1);
```

nvc_irq_disable

The description of nvc_irq_disable is shown as below:

Table 3-538. Function nvc_irq_disable

Function name	nvc_irq_disable
Function prototype	void nvc_irq_disable(uint8_t nvc_irq);
Function descriptions	disable NVIC request
Precondition	-
The called functions	-
Input parameter{in}	

nvic_irq	NVIC interrupt, refer to enum Enum IRQn_Type
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable window watchDog timer interrupt */
nvic_irq_disable(WWDGT_IRQn);
```

nvic_vector_table_set

The description of nvic_vector_table_set is shown as below:

Table 3-539. Function nvic_vector_table_set

Function name	nvic_vector_table_set
Function prototype	void nvic_vector_table_set(uint32_t nvic_vict_tab, uint32_t offset);
Function descriptions	set the NVIC vector table address
Precondition	-
The called functions	-
Input parameter{in}	
nvic_vict_tab	the RAM or FLASH base address
NVIC_VECTTAB_RAM	RAM base address
NVIC_VECTTAB_FLASH	Flash base address
Input parameter{in}	
offset	Vector Table offset (vector table start address= base address+offset)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set vector table address = NVIC_VECTTAB_FLASH +0x200 */
nvic_vector_table_set (NVIC_VECTTAB_FLASH,0x200);
```

system_lowpower_set

The description of system_lowpower_set is shown as below:

Table 3-540. Function system_lowpower_set

Function name	system_lowpower_set
Function prototype	void system_lowpower_set(uint8_t lowpower_mode);
Function descriptions	the state of the low power mode management

Precondition	-
The called functions	-
Input parameter{in}	
lowpower_mode	the low power mode state
<i>SCB_LPM_SLEEP_EXIT_T_ISR</i>	if chose this para, the system always enter low power mode by exiting from ISR
<i>SCB_LPM_DEEPSLEEP_P</i>	if chose this para, the system will enter the DEEPSLEEP mode
<i>SCB_LPM_WAKE_BY_ALL_INT</i>	if chose this para, the lowpower mode can be woke up by all the enable and disable interrupts
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* the system always enter low power mode by exiting from ISR */
system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);
```

system_lowpower_reset

The description of system_lowpower_reset is shown as below:

Table 3-541. Function system_lowpower_reset

Function name	system_lowpower_reset
Function prototype	void system_lowpower_reset(uint8_t lowpower_mode);
Function descriptions	the state of the low power mode management
Precondition	-
The called functions	-
Input parameter{in}	
lowpower_mode	the low power mode state
<i>SCB_LPM_SLEEP_EXIT_T_ISR</i>	if chose this para, the system will exit low power mode by exiting from ISR
<i>SCB_LPM_DEEPSLEEP_P</i>	if chose this para, the system will enter the SLEEP mode
<i>SCB_LPM_WAKE_BY_ALL_INT</i>	if chose this para, the lowpower mode only can be woke up by the enable interrupts
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* the system will exit low power mode by exiting from ISR */
```

```
system_lowpower_reset (SCB_LPM_SLEEP_EXIT_ISR);
```

systick_clksource_set

The description of systick_clksource_set is shown as below:

Table 3-542. Function systick_clksource_set

Function name	systick_clksource_set
Function prototype	void systick_clksource_set(uint32_t systick_clksource);
Function descriptions	set the systick clock source
Precondition	-
The called functions	-
Input parameter{in}	
systick_clksource	the systick clock source needed to choose
SYSTICK_CLKSOURC_E_HCLK	systick clock source is from HCLK
SYSTICK_CLKSOURC_E_HCLK_DIV8	systick clock source is from HCLK/8
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* systick clock source is HCLK/8 */
systick_clksource_set (SYSTICK_CLKSOURCE_HCLK_DIV8);
```

3.20. PMU

According to the Power management unit (PMU), provides three types of power saving modes, including Sleep, Deep-sleep and Standby mode. The PMU registers are listed in chapter [3.20.1](#), the PMU firmware functions are introduced in chapter [3.20.2](#).

3.20.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

Table 3-543. PMU Registers

Registers	Descriptions
PMU_CTL	Control register
PMU_CS	Control and status register

3.20.2. Descriptions of Peripheral functions

PMU firmware functions are listed in the table shown as below:

Table 3-544. PMU firmware function

Function name	Function description
pmu_deinit	deinitialize the PMU
pmu_lvd_select	select low voltage detector threshold
pmu_ldo_output_select	select LDO output voltage
pmu_lvd_disable	disable PMU lvd
pmu_highdriver_switch_select	high-driver mode switch
pmu_highdriver_mode_enable	high-driver mode enable
pmu_highdriver_mode_disable	high-driver mode disable
pmu_low_driver_mode_enable	low-driver mode enable in deep-sleep mode
pmu_lowdriver_lowpower_config	in deep-sleep mode, low-driver mode when use low power LDO
pmu_lowdriver_normalpower_config	in deep-sleep mode, low-driver mode when use normal power LDO
pmu_to_sleepmode	PMU work at sleep mode
pmu_to_deepsleepmode	PMU work at deepsleep mode
pmu_to_standbymode	pmu work at standby mode
pmu_wakeup_pin_enable	enable wakeup pin
pmu_backup_ldo_config	backup SRAM LDO on
pmu_backup_write_enable	backup domain write enable
pmu_wakeup_pin_disable	disable wakeup pin
pmu_backup_write_enable	enable backup domain write
pmu_backup_write_disable	disable backup domain write
pmu_flag_reset	reset flag bit
pmu_flag_get	get flag status

pmu_deinit

The description of pmu_deinit is shown as below:

Table 3-545. Function pmu_deinit

Function name	pmu_deinit
Function prototype	void pmu_deinit(void);
Function descriptions	deinitialize the PMU
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset PMU */
pmu_deinit();
```

pmu_lvd_select

The description of pmu_lvd_select is shown as below:

Table 3-546. Function pmu_lvd_select

Function name	pmu_lvd_select
Function prototype	void pmu_lvd_select(uint32_t lvdt_n);
Function descriptions	select low voltage detector threshold
Precondition	-
The called functions	-
Input parameter{in}	
lvdt_n	voltage threshold value
<i>PMU_LVDT_0</i>	voltage threshold is 2.2V
<i>PMU_LVDT_1</i>	voltage threshold is 2.3V
<i>PMU_LVDT_2</i>	voltage threshold is 2.4V

<i>PMU_LVDT_3</i>	voltage threshold is 2.5V
<i>PMU_LVDT_4</i>	voltage threshold is 2.6V
<i>PMU_LVDT_5</i>	voltage threshold is 2.7V
<i>PMU_LVDT_6</i>	voltage threshold is 2.8V
<i>PMU_LVDT_7</i>	voltage threshold is 2.9V
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select low voltage detector threshold as 3.1V */
pmu_lvd_select (PMU_LVDT_7);
```

pmu_ldo_output_select

The description of pmu_ldo_output_select is shown as below:

Table 3-547. Function pmu_ldo_output_select

Function name	pmu_ldo_output_select
Function prototype	void pmu_ldo_output_select(uint32_t ldo_output);
Function descriptions	select LDO output voltage
Precondition	-
The called functions	-
Input parameter{in}	
ldo_output	output voltage mode
<i>PMU_LDOVS_LOW</i>	low-driver mode enable in deep-sleep mode
<i>PMU_LDOVS_MID</i>	mid-driver mode disable in deep-sleep mode
<i>PMU_LDOVS_HIGH</i>	high-driver mode disable in deep-sleep mode
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* select output low-driver mode enable in deep-sleep mode */
pmu_ldo_output_select (PMU_LDOVS_LOW);
```

pmu_low_driver_mode_enable

The description of pmu_low_driver_mode_enable is shown as below:

Table 3-548. Function pmu_low_driver_mode_enable

Function name	pmu_low_driver_mode_enable
Function prototype	void pmu_low_driver_mode_enable(uint32_t lowdr_mode);
Function descriptions	enable low-driver mode in deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
lowdr_mode	enable low-driver mode or disable low-driver mode in deep-sleep mode
PMU_LOWDRI VER_E NABLE	enable low-driver mode in deep-sleep mode
PMU_LOWDRI VER_DI SABLE	disable low-driver mode in deep-sleep mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable low-driver mode in deep-sleep mode */
pmu_low_driver_mode_enable (PMU_LOWDRI  
VER_ENABLE);
```

pmu_highdriver_switch_select

The description of pmu_highdriver_switch_select is shown as below:

Table 3-549. Function pmu_highdriver_switch_select

Function name	pmu_highdriver_switch_select
Function prototype	void pmu_highdriver_switch_select(uint32_t highdr_switch);
Function descriptions	switch high-driver mode
Precondition	-
The called functions	-
Input parameter{in}	
highdr_switch	enable or disable high-driver mode switch
<i>PMU_HIGHDR_SWITC_H_NONE</i>	disable high-driver mode switch
<i>PMU_HIGHDR_SWITC_H_EN</i>	enable high-driver mode switch
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable high-driver mode switch */
pmu_highdriver_switch_select (PMU_HIGHDR_SWITCH_EN);
```

pmu_highdriver_mode_enable

The description of pmu_highdriver_mode_enable is shown as below:

Table 3-550. Function pmu_highdriver_mode_enable

Function name	pmu_highdriver_mode_enable
Function prototype	void pmu_highdriver_mode_enable(void);
Function descriptions	enable high-driver mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable high-driver mode */

pmu_highdriver_mode_enable ( );
```

pmu_highdriver_mode_disable

The description of pmu_highdriver_mode_disable is shown as below:

Table 3-551. Function pmu_highdriver_mode_disable

Function name	pmu_highdriver_mode_disable
Function prototype	void pmu_highdriver_mode_disable(void);
Function descriptions	disable high-driver mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable high-driver mode */

pmu_highdriver_mode_disable ( );
```

pmu_lvd_disable

The description of pmu_lvd_disable is shown as below:

Table 3-552. Function pmu_lvd_disable

Function name	pmu_lvd_disable
---------------	-----------------

Function prototype	void pmu_lvd_disable (void);
Function descriptions	disable PMU lvd
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable PMU lvd */
pmu_lvd_disable();
```

pmu_lowdriver_lowpower_config

The description of pmu_lowdriver_lowpower_config is shown as below:

Table 3-553. Function pmu_lowdriver_lowpower_config

Function name	pmu_lowdriver_lowpower_config
Function prototype	void pmu_lowdriver_lowpower_config(uint32_t mode);
Function descriptions	low-driver mode when use low power LDO
Precondition	-
The called functions	-
Input parameter{in}	
mode	Low power mode of LDO-
PMU_NORMALDR_LO_WPWR	normal driver when use low power LDO
PMU_LOWDR_LOWPWR	low-driver mode enabled when LDEN is 11 and use low power LDO
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* normal driver when use low power LDO */

pmu_lowdriver_lowpower_config (PMU_NORMALDR_LOWPWR);
```

pmu_lowdriver_normalpower_config

The description of pmu_lowdriver_normalpower_config is shown as below:

Table 3-554. pmu_lowdriver_normalpower_config

Function name	pmu_lowdriver_normalpower_config
Function prototype	void pmu_lowdriver_normalpower_config(uint32_t mode);
Function descriptions	low-driver mode when use normal power LDO
Precondition	-
The called functions	-
Input parameter{in}	
mode	Driver mode when use normal power LDO-
<i>PMU_NORMALDR_NORMALPWR</i>	normal driver when use normal power LDO
<i>PMU_LOWDR_NORMALPWR</i>	low-driver mode enabled when LDEN is 11 and use normal power LDO
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* normal driver when use normal power LDO */

pmu_lowdriver_normalpower_config (PMU_NORMALDR_NORMALPWR);
```

pmu_to_sleepmode

The description of pmu_to_sleepmode is shown as below:

Table 3-555. Function pmu_to_sleepmode

Function name	pmu_to_sleepmode
Function prototype	void pmu_to_sleepmode(uint8_t sleepmodecmd);
Function descriptions	PMU work at sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
sleepmodecmd	command to enter sleep mode
WFI_CMD	use WFI command
WFE_CMD	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* use WFI command to enter sleep mode */
pmu_to_sleepmode (WFI_CMD);
```

pmu_to_deepsleepmode

The description of pmu_to_deepsleepmode is shown as below:

Table 3-556. Function pmu_to_deepsleepmode

Function name	pmu_to_deepsleepmode
Function prototype	void pmu_to_deepsleepmode(uint32_t ldo,uint8_t deepsleepmodecmd);
Function descriptions	PMU work at deepsleep mode
Precondition	-
The called functions	-
Input parameter{in}	
ldo	ldo work mode
PMU_LDO_NORMAL	LDO normal work when pmu enter deepsleep mode

<i>PMU_LDO_LOWPOW ER</i>	LDO work at low power mode when pmu enter deepsleep mode
Input parameter{in}	
<i>deepsleepmodecmd</i>	command to enter deepsleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* use WFI command to enter deepsleep mode and LDO work at normal power */
pmu_to_deepsleepmode (PMU_LDO_NORMAL, WFI_CMD);
```

pmu_to_standbymode

The description of pmu_to_standbymode is shown as below:

Table 3-557. Function pmu_to_standbymode

Function name	pmu_to_standbymode
Function prototype	void pmu_to_standbymode(void);
Function descriptions	pmu work at standby mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* use WFI command to enter standby mode */
```

```
pmu_to_standbymode();
```

pmu_backup_ldo_config

The description of pmu_backup_ldo_config is shown as below:

Table 3-558. Function pmu_backup_ldo_config

Function name	pmu_backup_ldo_config
Function prototype	void pmu_backup_ldo_config(uint32_t bkp_ldo);
Function descriptions	backup SRAM LDO on
Precondition	-
The called functions	-
Input parameter{in}	
bkp_ldo	open or close the backup SRAM LDO
PMU_BLDOON_OFF	backup SRAM LDO closed
PMU_BLDOON_ON	open the backup SRAM LDO
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* backup SRAM LDO on */
pmu_backup_ldo_config (PMU_BLDOON_ON);
```

pmu_flag_reset

The description of pmu_flag_reset is shown as below:

Table 3-559. Function pmu_flag_reset

Function name	pmu_flag_reset
Function prototype	void pmu_flag_reset(uint32_t flag_reset);
Function descriptions	reset flag bit
Precondition	-

The called functions	-
Input parameter{in}	
flag_reset	flag
<i>PMU_FLAG_RESET_WAKEUP</i>	reset wakeup flag
<i>PMU_FLAG_RESET_STANDBY</i>	reset standby flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* reset wakeup flag */
pmu_flag_reset (PMU_FLAG_RESET_WAKEUP);
```

pmu_flag_get

The description of pmu_flag_get is shown as below:

Table 3-560. Function pmu_flag_get

Function name	pmu_flag_get
Function prototype	FlagStatus pmu_flag_get(uint32_t pmu_flag);
Function descriptions	get flag state
Precondition	-
The called functions	-
Input parameter{in}	
pmu_flag	flag
<i>PMU_FLAG_WAKEUP</i>	wakeup flag
<i>PMU_FLAG_STANDBY</i>	standby flag
<i>PMU_FLAG_LVD</i>	lvd flag
<i>PMU_FLAG_BLDORF</i>	backup SRAM LDO ready flag
<i>PMU_FLAG_LDOVSRF</i>	LDO voltage select ready flag

<i>PMU_FLAG_HDRF</i>	high-driver ready flag
<i>PMU_FLAG_HDSRF</i>	high-driver switch ready flag
<i>PMU_FLAG_LDRF</i>	low-driver mode ready flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get wakeup flag state*/
pmu_flag_get (PMU_FLAG_WAKEUP);
```

pmu_backup_write_enable

The description of pmu_backup_write_enable is shown as below:

Table 3-561. Function pmu_backup_write_enable

Function name	pmu_backup_write_enable
Function prototype	void pmu_backup_write_enable (void);
Function descriptions	enable backup domain write
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable backup domain write */
pmu_backup_write_enable ();
```

pmu_backup_write_disable

The description of pmu_backup_write_disable is shown as below:

Table 3-562. Function pmu_backup_write_disable

Function name	pmu_backup_write_disable
Function prototype	void pmu_backup_write_disable (void);
Function descriptions	disable backup domain write
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable backup domain write */
pmu_backup_write_disable();
```

pmu_wakeup_pin_enable

The description of pmu_wakeup_pin_enable is shown as below:

Table 3-563. Function pmu_wakeup_pin_enable

Function name	pmu_wakeup_pin_enable
Function prototype	void pmu_wakeup_pin_enable(void);
Function descriptions	enable wakeup pin
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable wakeup pin */
pmu_wakeup_pin_enable();
```

pmu_wakeup_pin_disable

The description of pmu_wakeup_pin_disable is shown as below:

Table 3-564. Function pmu_wakeup_pin_disable

Function name	pmu_wakeup_pin_disable
Function prototype	void pmu_wakeup_pin_disable (void);
Function descriptions	disable wakeup pin
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable wakeup pin */
pmu_wakeup_pin_disable();
```

3.21. RCU

RCU is the reset and clock unit. Reset Control includes the control of three kinds of reset: power reset, system reset and backup domain reset. The Clock Control unit provides a range of frequencies and clock functions. The RCU registers are listed in chapter [3.21.1](#), the RCU firmware functions are introduced in chapter [3.21.2](#).

3.21.1. Descriptions of Peripheral registers

RCU registers are listed in the table shown as below:

Table 3-565. RCU Registers

Registers	Descriptions
RCU_CTL	Control register
RCU_PLL	PLL register
RCU_CFG0	Clock configuration register 0
RCU_INT	Clock interrupt register
RCU_AHB1RST	AHB1 reset register
RCU_AHB2RST	AHB2 reset register
RCU_AHB3RST	AHB3 reset register
RCU_APB1RST	APB1 reset register
RCU_APB2RST	APB2 reset register
RCU_AHB1EN	AHB1 enable register
RCU_AHB2EN	AHB2 enable register
RCU_AHB3EN	AHB3 enable register
RCU_APB1EN	APB1 enable register
RCU_APB2EN	APB2 enable register
RCU_AHB1SPEN	AHB1 sleep mode enable register
RCU_AHB2SPEN	AHB2 sleep mode enable register
RCU_AHB3SPEN	AHB3 sleep mode enable register
RCU_APB1SPEN	APB1 sleep mode enable register
RCU_APB2SPEN	APB2 sleep mode enable register
RCU_BDCTL	Backup domain control register
RCU_RSTSCK	Reset source/clock register
RCU_PLLSSCTL	PLL clock spread spectrum control register
RCU_PLLI2S	PLL I2S register
RCU_PLLSAI	PLLSAI register

Registers	Descriptions
RCU_CFG1	Configuration register 1
RCU_ADDCTL	Additional clock control register
RCU_ADDINT	Additional clock interrupt register
RCU_ADDAPB1RST	APB1 additional reset register
RCU_ADDAPB1EN	APB1 additional enable register
RCU_ADDAPB1SPEN	APB1 additional sleep mode enable register
RCU_VKEY	voltage key register
RCU_DSV	Deep-sleep mode voltage register

3.21.2. Descriptions of Peripheral functions

Table 3-566. RCU firmware function

Function name	Function description
rcu_deinit	deinitialize the RCU
rcu_periph_clock_enable	enable the peripherals clock
rcu_periph_clock_disable	disable the peripherals clock
rcu_periph_clock_sleep_enable	enable the peripherals clock when in sleep mode
rcu_periph_clock_sleep_disable	disable the peripherals clock when in sleep mode
rcu_periph_reset_enable	enable the peripherals reset
rcu_periph_reset_disable	disable the peripheral reset
rcu_bkp_reset_enable	enable the BKP domain reset
rcu_bkp_reset_disable	disable the BKP domain reset
rcu_system_clock_source_config	configure the system clock source
rcu_system_clock_source_get	get the system clock source
rcu_ahb_clock_config	configure the AHB clock prescaler selection
rcu_apb1_clock_config	configure the APB1 clock prescaler selection
rcu_apb2_clock_config	configure the APB2 clock prescaler selection
rcu_ckout0_config	configure the CK_OUT0 clock source
rcu_ckout1_config	configure the CK_OUT1 clock source

Function name	Function description
rcu_pll_config	configure the main PLL clock
rcu_pll2s_config	configure the PLLI2S clock
rcu_pllsai_config	configure the PLLSAI clock
rcu_RTC_clock_config	configure the RTC clock source selection
rcu_I2S_clock_config	configure the I2S clock source selection
rcu_ck48m_clock_config	configure the CK48M clock selection
rcu_pll48m_clock_config	configure the PLL48M clock selection
rcu_timer_clock_prescaler_config	configure the TIMER clock prescaler selection
rcu_tli_clock_div_config	configure the TLI clock division selection
rcu_flag_get	get the clock stabilization and peripheral reset flags
rcu_all_reset_flag_clear	clear all the reset flag
rcu_interrupt_flag_get	get the clock stabilization interrupt and ckm flags
rcu_interrupt_flag_clear	clear the interrupt flags
rcu_interrupt_enable	enable the stabilization interrupt
rcu_interrupt_disable	disable the stabilization interrupt
rcu_lxtal_drive_capability_config	configure the LXTAL drive capability
rcu_osc_stab_wait	wait for oscillator stabilization flags is SET or oscillator startup is timeout
rcu_osc_on	turn on the oscillator
rcu_osc_off	turn off the oscillator
rcu_osc_bypass_mode_enable	enable the oscillator bypass mode
rcu_osc_bypass_mode_disable	disable the oscillator bypass mode
rcu_hxtal_clock_monitor_enable	enable the HXTAL clock monitor
rcu_hxtal_clock_monitor_disable	disable the HXTAL clock monitor
rcu_iric16m_adjust_value_set	set the IRC8M adjust value
rcu_spread_spectrum_config	configure the spread spectrum modulation for the main PLL clock
rcu_spread_spectrum_enable	enable the spread spectrum modulation for the main PLL

Function name	Function description
	clock
rcu_spread_spectrum_disable	disable the spread spectrum modulation for the main PLL clock
rcu_deepsleep_voltage_set	set the deep-sleep mode voltage value
rcu_voltage_key_unlock	unlock the voltage key
rcu_clock_freq_get	get the system clock, bus clock frequency

Enum rcu_periph_enum
Table 3-567. Enum rcu_periph_enum

enum name	Function description
RCU_GPIOx	GPIO ports clock (x=A,B,C,D,E,F,G ,H,I)
RCU_CRC	CRC clock
RCU_BKPSRAM	BKPSRAM clock
RCU_TCMSRAM	TCMSRAM clock
RCU_DMAX	DMAx clock (x=0,1)
RCU_IPA	IPA clock
RCU_ENET	ENET clock
RCU_ENETTX	ENETTX clock
RCU_ENETRX	ENETRX clock
RCU_ENETPTP	ENETPTP clock
RCU_USBHS	USBHS clock
RCU_USBHSULPI	USBHSULPI clock
RCU_DCI	DCI clock
RCU_TRNG	TRNG clock
RCU_USBFS	USBFS clock
RCU_EXMC	EXMC clock
RCU_TIMERx	TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
RCU_WWDGT	WWDGT clock
RCU_SPIx	SPIx clock (x=0,1,2,3,4,5)
RCU_USARTx	USARTx clock (x=0,1,2,5)
RCU_UARTx	UARTx clock (x=3,4 ,6,7)
RCU_I2Cx	I2Cx clock (x=0,1,2)
RCU_CANx	CANx clock (x=0,1)
RCU_PMU	PMU clock
RCU_DAC	DAC clock
RCU_RTC	RTC clock
RCU_ADCx	ADCx clock (x=0,1,2)
RCU_SDIO	SDIO clock

enum name	Function description
RCU_SYSCFG	SYSCFG interface clock
RCU_TLI	TLI clock
RCU_CTC	CTC clock
RCU_IREF	IREF clock

Enum rcu_periph_sleep_enum
Table 3-568. Enum rcu_periph_sleep_enum

enum name	Function description
RCU_GPIOx_SLP	GPIO clock (x=A,B,C,D,E,F,G ,H,I)
RCU_CRC_SLP	CRC clock
RCU_FMC_SLP	FMC clock
RCU_SRAM1_SLP	SRAM0 clock
RCU_SRAM1_SLP	SRAM1 clock
RCU_BKPSRAM	BKPSRAM clock
RCU_SRAM2_SLP	SRAM2 clock
RCU_DMAx_SLP	DMAx clock (x=0,1)
RCU_IPA_SLP	IPA clock
RCU_ENET_SLP	ENET clock
RCU_ENETTX_SLP	ENETTX clock
RCU_ENETRX_SLP	ENETRX clock
RCU_ENETPTP_SLP	ENETPTP clock
RCU_USBHS_SLP	USBHS clock
RCU_USBHSULPI_SLP P	USBHSULPI clock
RCU_DCI_SLP	DCI clock
RCU_TRNG_SLP	TRNG clock
RCU_USBFS_SLP	USBFS clock
RCU_EXMC_SLP	EXMC clock
RCU_TIMERx_SLP	TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
RCU_WWDGT_SLP	WWDGT clock
RCU_SPIx_SLP	SPIx clock (x=0,1,2,3,4,5)
RCU_USARTx_SLP	USARTx clock (x=0,1,2,5)
RCU_UARTx_SLP	UARTx clock (x=3,4 ,6,7)
RCU_I2Cx_SLP	I2Cx clock (x=0,1,2)
RCU_CANx_SLP	CANx clock (x=0,1)
RCU_PMU_SLP	PMU clock
RCU_DAC_SLP	DAC clock
RCU_RTC_SLP	RTC clock
RCU_ADCx_SLP	ADCx clock (x=0,1,2)
RCU_SDIO_SLP	SDIO clock
RCU_SYSCFG_SLP	SYSCFG interfaceclock

enum name	Function description
RCU_TLI_SLP	TLI clock
RCU_CTC_SLP	CTC clock
RCU_IREF_SLP	IREF clock

Enum rcu_periph_reset_enum

Table 3-569. Enum rcu_periph_reset_enum

enum name	Function description
RCU_GPIOxRST	reset GPIOx clock (x=A,B,C,D,E,F,G,H,I)
RCU_CRCRST	reset CRC clock
RCU_DMAMRST	reset DMAx clock (x=0,1)
RCU_IPARST	reset IPA clock
RCU_ENETRST	reset ENET clock
RCU_USBHSRST	reset USBHSclock
RCU_DCIRST	reset DCI clock
RCU_TRNGRST	reset TRNG clock
RCU_USBFSRST	reset USBFS clock
RCU_EXMCRST	reset EXMC clock
RCU_TIMERxRST	reset TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
RCU_WWDGTRST	reset WWDGT clock
RCU_SPIxRST	reset SPIx clock (x=0,1,2,3,4,5)
RCU_USARTxRST	reset USARTx clock (x=0,1,2,5)
RCU_UARTxRST	reset UARTx clock (x=3,4,6,7)
RCU_I2CxRST	reset I2Cx clock (x=0,1,2)
RCU_CANxRST	reset CANx clock (x=0,1)
RCU_PMURST	reset PMU clock
RCU_DACRST	reset DAC clock
RCU_ADCxRST	reset ADCx clock (x=0,1,2)
RCU_SDIORST	reset SDIO clock
RCU_SYSCFGGRST	reset SYSCFG clock
RCU_TLIRST	reset TLI clock
RCU_CTCRST	reset CAU clock
RCU_IREFRST	reset IREF clock

Enum rcu_flag_enum

Table 3-570. Enum rcu_flag_enum

enum name	Function description
RCU_FLAG_IRC16MS_TB	IRC16M stabilization flag
RCU_FLAG_HXTALST_B	HXTAL stabilization flag

enum name	Function description
RCU_FLAG_PLLI2SST_B	PLL2S stabilization flag
RCU_FLAG_PLLSAIST_B	PLLSAI stabilization flag
RCU_FLAG_PLLSTB	PLL stabilization flag
RCU_FLAG_LXTALST_B	LXTAL stabilization flag
RCU_FLAG_IRC32KST_B	IRC32K stabilization flag
RCU_FLAG_IRC48MS_TB	IRC48M stabilization flag
RCU_FLAG_EPRST	external PIN reset flag
RCU_FLAG_PORRST	power reset flag
RCU_FLAG_BORRST	BOR reset flag
RCU_FLAG_SWRST	software reset flag
RCU_FLAG_FWDGTR_ST	free watchdog timer reset flag
RCU_FLAG_WWDGTR_ST	window watchdog timer reset flag
RCU_FLAG_LPRST	low-power reset flag

Enum rcu_int_flag_enum

Table 3-571. Enum rcu_int_flag_enum

enum name	Function description
RCU_INT_FLAG_IRC32KSTB	IRC32K stabilization interrupt flag
RCU_INT_FLAG_LXTALSTB	LXTAL stabilization interrupt flag
RCU_INT_FLAG_IRC16MSTB	IRC16M stabilization interrupt flag
RCU_INT_FLAG_HXTALSTB	HXTAL stabilization interrupt flag
RCU_INT_FLAG_PLLSTB	PLL stabilization interrupt flag
RCU_INT_FLAG_PLLI2SSTB	PLL2S stabilization interrupt flag
RCU_INT_FLAG_PLLSAISTB	PLLSAI stabilization interrupt flag
RCU_INT_FLAG_CKM	HXTAL clock stuck interrupt flag
RCU_INT_FLAG_IRC48MSTB	IRC48M stabilization interrupt flag

Enum rcu_int_flag_clear_enum

Table 3-572. Enum rcu_int_flag_clear_enum

enum name	Function description
RCU_INT_FLAG_IRC3_2KSTB_CLR	IRC32K stabilization interrupt flag clear
RCU_INT_FLAG_LXTA_LSTB_CLR	LXTAL stabilization interrupt flag clear
RCU_INT_FLAG_IRC1_6MSTB_CLR	IRC16M stabilization interrupt flag clear
RCU_INT_FLAG_HXTA_LSTB_CLR	HXTAL stabilization interrupt flag clear
RCU_INT_FLAG_PLLS_TB_CLR	PLL stabilization interrupt flag clear
RCU_INT_FLAG_PLLI2_SSTB_CLR	PLL2S stabilization interrupt flag clear
RCU_INT_FLAG_PLLS_AISTB_CLR	PLLSAI stabilization interrupt flag clear
RCU_INT_FLAG_CKM_CLR	clock stuck interrupt flag clear
RCU_INT_FLAG_IRC4_8MSTB_CLR	IRC48M stabilization interrupt flag clear

Enum rcu_int_enum

Table 3-573. Enum rcu_int_enum

enum name	Function description
RCU_INT_IRC32KSTB	IRC32K stabilization interrupt enable
RCU_INT_LXTALSTB	LXTAL stabilization interrupt enable
RCU_INT_IRC16MSTB	IRC16M stabilization interrupt enable
RCU_INT_IRC48MSTB	IRC48M stabilization interrupt enable
RCU_INT_HXTALSTB	HXTAL stabilization interrupt enable
RCU_INT_PLLSTB	PLL stabilization interrupt enable
RCU_INT_PLLI2SSTB	PLL2S stabilization interrupt enable
RCU_INT_PLLSAISTB	PLLSAI stabilization interrupt enable

Enum rcu_osc_type_enum

Table 3-574. Enum rcu_osc_type_enum

enum name	Function description
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)
RCU_IRC16M	internal 16M RC oscillators(IRC16M)
RCU_IRC48M	internal 48M RC oscillators(IRC48M)

enum name	Function description
RCU_IRC32K	internal 32K RC oscillator(IRC32K)
RCU_PLL_CK	phase locked loop(PLL)
RCU_PLLI2S_CK	PLL2S phase locked loop
RCU_PLLSAI_CK	PLLSAI phase locked loop

Enum rcu_clock_freq_enum

Table 3-575. Enum rcu_clock_freq_enum

enum name	Function description
CK_SYS	system clock frequency
CK_AHB	AHB clock frequency
CK_APB1	APB1 clock frequency
CK_APB2	APB2 clock frequency

rcu_deinit

The description of rcu_deinit is shown as below:

Table 3-576. Function rcu_deinit

Function name	rcu_deinit
Function prototype	void rcu_deinit(void);
Function descriptions	deinitialize the RCU
Precondition	-
The called functions	rcu_osc_stab_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset RCU */
rcu_deinit();
```

rcu_periph_clock_enable

The description of rcu_periph_clock_enable is shown as below:

Table 3-577. Function rcu_periph_clock_enable

Function name	rcu_periph_clock_enable
Function prototype	void rcu_periph_clock_enable(rcu_periph_enum periph);
Function descriptions	enable the peripherals clock
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to Table 3-567. Enum rcu_periph_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the USART0 clock */
rcu_periph_clock_enable(RCU_USART0);
```

rcu_periph_clock_disable

The description of rcu_periph_clock_disable is shown as below:

Table 3-578. Function rcu_periph_clock_disable

Function name	rcu_periph_clock_disable
Function prototype	void rcu_periph_clock_disable(rcu_periph_enum periph);
Function descriptions	disable the peripherals clock
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to Table 3-567. Enum rcu_periph_enum
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable the USART0 clock */

rcu_periph_clock_disable(RCU_USART0);
```

rcu_periph_clock_sleep_enable

The description of `rcu_periph_clock_sleep_enable` is shown as below:

Table 3-579. Function `rcu_periph_clock_sleep_enable`

Function name	rcu_periph_clock_sleep_enable
Function prototype	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
Function descriptions	enable the peripherals clock when in sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to Table 3-568. Enum <code>rcu_periph_sleep_enum</code>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CRC clock when in sleep mode */

rcu_periph_clock_sleep_enable(RCU_CRC_SLP);
```

rcu_periph_clock_sleep_disable

The description of `rcu_periph_clock_sleep_disable` is shown as below:

Table 3-580. Function `rcu_periph_clock_sleep_disable`

Function name	rcu_periph_clock_sleep_disable
Function prototype	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);

Function descriptions		disable the peripherals clock when in sleep mode
Precondition		-
The called functions		-
Input parameter{in}		
periph	RCU peripherals, refer to Table 3-568. Enum rcu_periph_sleep_enum	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

```
/* disable the CRC clock when in sleep mode */
rcu_periph_clock_sleep_disable(RCU_CRC_SLP);
```

rcu_periph_reset_enable

The description of `rcu_periph_reset_enable` is shown as below:

Table 3-581. Function `rcu_periph_reset_enable`

Function name		rcu_periph_reset_enable
Function prototype		void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
Function descriptions		enable the peripherals reset
Precondition		-
The called functions		-
Input parameter{in}		
periph_reset	RCU peripherals reset, refer to Table 3-569. Enum rcu_periph_reset_enum	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

```
/* enable SPI0 reset */
rcu_periph_reset_enable(RCU_SPI0RST);
```

rcu_periph_reset_disable

The description of `rcu_periph_reset_disable` is shown as below:

Table 3-582. Function `rcu_periph_reset_disable`

Function name	rcu_periph_reset_disable
Function prototype	void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);
Function descriptions	disable the peripheral reset
Precondition	-
The called functions	-
Input parameter{in}	
periph_reset	RCU peripherals reset, refer to Table 3-569. Enum rcu_periph_reset_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 reset */
rcu_periph_reset_disable(RCU_SPI0RST);
```

rcu_bkp_reset_enable

The description of `rcu_bkp_reset_enable` is shown as below:

Table 3-583. Function `rcu_bkp_reset_enable`

Function name	rcu_bkp_reset_enable
Function prototype	void rcu_bkp_reset_enable(void);
Function descriptions	enable the BKP domain reset
Precondition	-
The called functions	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the BKP domain */

rcu_bkp_reset_enable();
```

rcu_bkp_reset_disable

The description of `rcu_bkp_reset_disable` is shown as below:

Table 3-584. Function `rcu_bkp_reset_disable`

Function name	rcu_bkp_reset_disable
Function prototype	void rcu_bkp_reset_disable(void);
Function descriptions	disable the BKP domain reset
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the BKP domain reset */

rcu_bkp_reset_disable();
```

rcu_system_clock_source_config

The description of `rcu_system_clock_source_config` is shown as below:

Table 3-585. Function rcu_system_clock_source_config

Function name	rcu_system_clock_source_config
Function prototype	void rcu_system_clock_source_config(uint32_t ck_sys);
Function descriptions	configure the system clock source
Precondition	-
The called functions	-
Input parameter{in}	
ck_sys	system clock source select
<i>RCU_CKSYSRC_IRC 16M</i>	select CK_IRC16M as the CK_SYS source
<i>RCU_CKSYSRC_HXTAL</i>	select CK_HXTAL as the CK_SYS source
<i>RCU_CKSYSRC_PLL P</i>	select CK_PLLP as the CK_SYS source
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure the CK_HXTAL as the CK_SYS source */
rcu_system_clock_source_config(RCU_CKSYSRC_HXTAL);

```

rcu_system_clock_source_get

The description of rcu_system_clock_source_get is shown as below:

Table 3-586. Function rcu_system_clock_source_get

Function name	rcu_system_clock_source_get
Function prototype	uint32_t rcu_system_clock_source_get(void);
Function descriptions	get the system clock source
Precondition	-
The called functions	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
<i>uint32_t</i>	which clock is selected as CK_SYS source
<i>RCU_SCSS_IRC16M</i>	CK_IRC16M is selected as the CK_SYS source
<i>RCU_SCSS_HXTAL</i>	CK_HXTAL is selected as the CK_SYS source
<i>RCU_SCSS_PLLP</i>	CK_PLLP is selected as the CK_SYS source

Example:

```
uint32_t temp_cksys_status;  
/* get the CK_SYS source */  
temp_cksys_status = rcu_system_clock_source_get();
```

rcu_ahb_clock_config

The description of **rcu_ahb_clock_config** is shown as below:

Table 3-587. Function *rcu_ahb_clock_config*

Function name	<i>rcu_ahb_clock_config</i>
Function prototype	void <i>rcu_ahb_clock_config(uint32_t ck_ahb);</i>
Function descriptions	configure the AHB clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
ck_ahb	AHB clock prescaler selection
<i>RCU_AHB_CKSYS_DI_Vx</i>	select CK_SYS / x, (x=1, 2, 4, 8, 16, 64, 128, 256, 512)
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure CK_SYS/128 */
rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

rcu_apb1_clock_config

The description of rcu_apb1_clock_config is shown as below:

Table 3-588. Function rcu_apb1_clock_config

Function name	rcu_apb1_clock_config
Function prototype	void rcu_apb1_clock_config(uint32_t ck_apb1);
Function descriptions	configure the APB1 clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
ck_apb1	APB1 clock prescaler selection
<i>RCU_APB1_CKAHB_D</i> <i>IV1</i>	select CK_AHB as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV2</i>	select CK_AHB/2 as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV4</i>	select CK_AHB/4 as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV8</i>	select CK_AHB/8 as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV16</i>	select CK_AHB/16 as CK_APB1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CK_AHB/16 as CK_APB1 */
```

```
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);
```

rcu_apb2_clock_config

The description of rcu_apb2_clock_config is shown as below:

Table 3-589. Function rcu_apb2_clock_config

Function name	rcu_apb2_clock_config
Function prototype	void rcu_apb2_clock_config(uint32_t ck_apb2);
Function descriptions	configure the APB2 clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
ck_apb2	APB2 clock prescaler selection
<i>RCU_APB2_CKAHB_DIV1</i>	select CK_AHB as CK_APB2
<i>RCU_APB2_CKAHB_DIV2</i>	select CK_AHB/2 as CK_APB2
<i>RCU_APB2_CKAHB_DIV4</i>	select CK_AHB/4 as CK_APB2
<i>RCU_APB2_CKAHB_DIV8</i>	select CK_AHB/8 as CK_APB2
<i>RCU_APB2_CKAHB_DIV16</i>	select CK_AHB/16 as CK_APB2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CK_AHB/8 as CK_APB2 */
rcu_apb2_clock_config(RCU_APB2_CKAHB_DIV8);
```

rcu_ckout0_config

The description of rcu_ckout0_config is shown as below:

Table 3-590. Function rcu_ckout0_config

Function name	rcu_ckout0_config
Function prototype	void rcu_ckout0_config(uint32_t ckout0_src, uint32_t ckout0_div);
Function descriptions	configure the CK_OUT0 clock source
Precondition	-
The called functions	-
Input parameter{in}	
ckout0_src	CK_OUT0 clock source selection
<i>RCU_CKOUT0SRC_IR C16M</i>	select IRC16M clock
<i>RCU_CKOUT0SRC_LX TAL</i>	select LXTAL clock
<i>RCU_CKOUT0SRC_H XTAL</i>	select HXTAL clock
<i>RCU_CKOUT0SRC_C KPLL_PLLP</i>	select PLLP clock
Input parameter{in}	
ckout0_div	CK_OUT0 divider
<i>RCU_CKOUT0_DIVx(x =1,2,3,4,5)</i>	CK_OUT0 is divided by x
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the HXTAL as CK_OUT0 clock source */
rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL, RCU_CKOUT0_DIV1);
```

rcu_ckout1_config

The description of rcu_ckout1_config is shown as below:

Table 3-591. Function rcu_ckout1_config

Function name	rcu_ckout1_config
Function prototype	void rcu_ckout1_config(uint32_t ckout1_src, uint32_t ckout1_div);
Function descriptions	configure the CK_OUT1 clock source and divider
Precondition	-
The called functions	-
Input parameter{in}	
ckout1_src	CK_OUT1 clock source selection
<i>RCU_CKOUT1SRC_S YSTEMCLOCK</i>	select system clock
<i>RCU_CKOUT1SRC_PL L2SR</i>	select PLLI2SR clock
<i>RCU_CKOUT1SRC_H XTAL</i>	select HXTAL clock
<i>RCU_CKOUT1SRC_PL LP</i>	select PLLP clock
Input parameter{in}	
ckout1_div	CK_OUT1 divider
<i>RCU_CKOUT1_DIVx(x =1,2,3,4,5)</i>	CK_OUT1 is divided by x
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure the HXTAL as CK_OUT1 clock source */

rcu_ckout1_config(RCU_CKOUT1SRC_HXTAL, RCU_CKOUT1_DIV1);

```

rcu_pll_config

The description of rcu_pll_config is shown as below:

Table 3-592. Function rcu_pll_config

Function name	rcu_pll_config
Function prototype	ErrStatus rcu_pll_config(uint32_t pll_src, uint32_t pll_psc, uint32_t pll_n, uint32_t pll_p, uint32_t pll_q);
Function descriptions	configure the main PLL clock
Precondition	-
The called functions	-
Input parameter{in}	
pll_src	PLL clock source selection
<i>RCU_PLLSRC_IRC16M</i>	IRC16M clock is selected as source clock of PLL,PLLSAI,PLLI2S
<i>RCU_PLLSRC_HXTAL</i>	HXTAL is selected as source clock of PLL,PLLSAI,PLLI2S
Input parameter{in}	
pll_psc	the PLL VCO source clock prescaler
<i>uint32_t</i>	2~63
Input parameter{in}	
pll_n	the PLL VCO clock multi factor
<i>uint32_t</i>	64~500
Input parameter{in}	
pll_p	the PLLP output frequency division factor from PLL VCO clock
<i>uint32_t</i>	2,4,6,8
Input parameter{in}	
pll_q	the PLL Q output frequency division factor from PLL VCO clock
<i>uint32_t</i>	2~15
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* Configure the main PLL, PSC = 8, PLL_N = 240, PLL_P = 2, PLL_Q = 5 */
```

```
rcu_pll_config(RCU_PLLSRC_HXTAL,8,240,2,5);
```

rcu_plli2s_config

The description of rcu_plli2s_config is shown as below:

Table 3-593. Function rcu_plli2s_config

Function name	rcu_plli2s_config
Function prototype	ErrStatus rcu_plli2s_config(uint32_t plli2s_n, uint32_t plli2s_r);
Function descriptions	configure the PLLI2S clock
Precondition	-
The called functions	-
Input parameter{in}	
plli2s_n	the PLLI2S VCO clock multi factor
uint32_t	50~500
Input parameter{in}	
plli2s_r	the PLLI2S R output frequency division factor from PLLI2S VCO clock
uint32_t	2~7
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* configure the PLLI2S n = 100, PLLI2S r =2*/
rcu_plli2s_config (100, 2);
```

rcu_pll sai config

The description of rcu_pll sai config is shown as below:

Table 3-594. Function rcu_pll sai config

Function name	rcu_pll sai config
Function prototype	ErrStatus rcu_pll sai config(uint32_t pll sai_n, uint32_t pll sai_p, uint32_t

	pllsai_r);
Function descriptions	configure the PLLSAI clock
Precondition	-
The called functions	-
Input parameter{in}	
pllsai_n	the PLLSAI VCO clock multi factor
<i>uint32_t</i>	50~500
Input parameter{in}	
pllsai_p	the PLLSAI P output frequency division factor from PLL VCO clock
<i>uint32_t</i>	2~15
Input parameter{in}	
pllsai_r	the PLLSAI R output frequency division factor from PLL VCO clock
<i>uint32_t</i>	2~7
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* configure the PLLSAI n = 100, PLLSAI p = 2, PLLSAI r = 2 */
rcu_pllsai_config (100, 2,2);
```

rcu_rtc_clock_config

The description of **rcu_rtc_clock_config** is shown as below:

Table 3-595. Function **rcu_rtc_clock_config**

Function name	rcu_rtc_clock_config
Function prototype	void rcu_rtc_clock_config(uint32_t rtc_clock_source);
Function descriptions	configure the RTC clock source selection
Precondition	-
The called functions	-

Input parameter{in}	
rtc_clock_source	RTC clock source selection
<i>RCU_RTCSRC_NONE</i>	no clock selected
<i>RCU_RTCSRC_LXTAL</i>	select CK_LXTAL as RTC source clock
<i>RCU_RTCSRC_IRC32K</i>	select CK_IRC32K as RTC source clock
<i>RCU_RTCSRC_HXTAL_DIV_RTCDIV</i>	select CK_HXTAL or RTCDIV as RTC source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the RTC clock source selection */

rcu_rtc_clock_config(RCU_RTCSRC_IRC32K);
```

rcu_i2s_clock_config

The description of **rcu_i2s_clock_config** is shown as below:

Table 3-596. Function **rcu_i2s_clock_config**

Function name	rcu_i2s_clock_config
Function prototype	void rcu_i2s_clock_config(uint32_t i2s_clock_source);
Function descriptions	configure the I2S clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
i2s_clock_source	I2S clock source selection
<i>RCU_I2SSRC_PLLI2S</i>	CK_PLLI2S selected as I2S source clock
<i>RCU_I2SSRC_I2S_CKIN</i>	external i2s_ckin pin selected as I2S source clock
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* configure the I2S clock source selection */
rcu_i2s_clock_config(RCU_I2SSRC_PLLI2S);
```

rcu_ck48m_clock_config

The description of **rcu_ck48m_clock_config** is shown as below:

Table 3-597. Function **rcu_ck48m_clock_config**

Function name	rcu_ck48m_clock_config
Function prototype	void rcu_ck48m_clock_config(uint32_t ck48m_clock_source);
Function descriptions	configure the CK48M clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
ck48m_clock_source	CK48M clock source selection
<i>RCU_CK48MSRC_PLL48M</i>	CK_PLL48M selected as CK48M source clock
<i>RCU_CK48MSRC_IRC48M</i>	CK_IRC48M selected as CK48M source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the CK48M clock source selection */
rcu_ck48m_clock_config (RCU_CK48MSRC_IRC48M);
```

rcu_pll48m_clock_config

The description of `rcu_pll48m_clock_config` is shown as below:

Table 3-598. Function `rcu_pll48m_clock_config`

Function name	<code>rcu_pll48m_clock_config</code>
Function prototype	<code>void rcu_pll48m_clock_config(uint32_t pll48m_clock_source);</code>
Function descriptions	configure the PLL48M clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
<code>pll48m_clock_source</code>	PLL48M clock source selection
<code>RCU_PLL48MSRC_PLQ</code>	CK_PLLQ selected as PLL48M source clock
<code>RCU_PLL48MSRC_PLSAIP</code>	CK_PLLSAIP selected as PLL48M source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLL48M clock selection */
rcu_pll48m_clock_config(RCU_PLL48MSRC_PLLQ);
```

rcu_timer_clock_prescaler_config

The description of `rcu_timer_clock_prescaler_config` is shown as below:

Table 3-599. Function `rcu_timer_clock_prescaler_config`

Function name	<code>rcu_timer_clock_prescaler_config</code>
Function prototype	<code>void rcu_timer_clock_prescaler_config(uint32_t timer_clock_prescaler);</code>
Function descriptions	configure the TIMER clock prescaler selection
Precondition	-
The called functions	-

Input parameter{in}	
timer_clock_prescaler	TIMER时钟源选择
<i>RCU_TIMER_PSC_MU_L2</i>	If CK_APBx = CK_AHB or CK_APBx = CK_AHB/2, CK_TIMERx = CK_AHB, else CK_TIMERx = 2 x CK_APBx
<i>RCU_TIMER_PSC_MU_L4</i>	If CK_APBx = CK_AHB or CK_APBx = CK_AHB/2 or CK_APBx = CK_AHB/4, CK_TIMERx = CK_AHB, else CK_TIMERx = 4 x CK_APBx
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER clock source */
rcu_timer_clock_prescaler_config(RCU_TIMER_PSC_MUL4);
```

rcu_tli_clock_div_config

The description of **rcu_tli_clock_div_config** is shown as below:

Table 3-600. Function rcu_tli_clock_div_config

Function name	rcu_tli_clock_div_config
Function prototype	void rcu_tli_clock_div_config(uint32_t pllsai_r_div);
Function descriptions	configure the PLLSAIR divider used as input of TLI
Precondition	-
The called functions	-
Input parameter{in}	
pllsai_r_div	PLLSAIR divider used as input of TLI
<i>RCU_PLLSAIR_DIV2</i>	PLLSAIR/2
<i>RCU_PLLSAIR_DIV4</i>	PLLSAIR/4
<i>RCU_PLLSAIR_DIV8</i>	PLLSAIR/8
<i>RCU_PLLSAIR_DIV16</i>	PLLSAIR/16
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure the TLI prescaler factor from PLLSAIR clock */
rcu_tli_clock_div_config (RCU_PLLSAIR_DIV4);
```

rcu_flag_get

The description of `rcu_flag_get` is shown as below:

Table 3-601. Function `rcu_flag_get`

Function name	rcu_flag_get
Function prototype	FlagStatus rcu_flag_get(rcu_flag_enum flag);
Function descriptions	get the clock stabilization and periphral reset flags
Precondition	-
The called functions	-
Input parameter{in}	
flag	the clock stabilization and periphral reset flags, refer to Table 3-570. Enum rcu_flag_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the clock stabilization flag */

if(RESET != rcu_flag_get(RCU_FLAG_LXTALSTB)){
}
```

rcu_all_reset_flag_clear

The description of `rcu_all_reset_flag_clear` is shown as below:

Table 3-602. Function `rcu_all_reset_flag_clear`

Function name	rcu_all_reset_flag_clear
----------------------	--------------------------

Function prototype	void rcu_all_reset_flag_clear(void);
Function descriptions	clear all the reset flag
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear all the reset flag */
rcu_all_reset_flag_clear();
```

rcu_interrupt_flag_get

The description of `rcu_interrupt_flag_get` is shown as below:

Table 3-603. Function `rcu_interrupt_flag_get`

Function name	rcu_interrupt_flag_get
Function prototype	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);
Function descriptions	get the clock stabilization interrupt and ckm flags
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	interrupt and ckm flags, refer to Table 3-571. Enum <code>rcu_int_flag_enum</code>
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```

/* get the clock stabilization interrupt flag */

if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){

}

```

rcu_interrupt_flag_clear

The description of `rcu_interrupt_flag_clear` is shown as below:

Table 3-604. Function `rcu_interrupt_flag_clear`

Function name	rcu_interrupt_flag_clear
Function prototype	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag_clear)
Function descriptions	clear the interrupt flags
Precondition	-
The called functions	-
Input parameter{in}	
int_flag_clear	clock stabilization and stuck interrupt flags clear, refer to Table 3-572. Enum rcu_int_flag_clear enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* clear the interrupt HXTAL stabilization interrupt flag */

rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);

```

rcu_interrupt_enable

The description of `rcu_interrupt_enable` is shown as below:

Table 3-605. Function `rcu_interrupt_enable`

Function name	rcu_interrupt_enable
Function prototype	void rcu_interrupt_enable(rcu_int_enum stab_int);
Function descriptions	enable the stabilization interrupt
Precondition	-

The called functions		-
Input parameter{in}		
stab_int	clock stabilization interrupt, refer to Table 3-573. Enum rcu_int_enum	
Output parameter{out}		
-	-	-
Return value		
-	-	-

Example:

```
/* enable the HXTAL stabilization interrupt */
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

rcu_interrupt_disable

The description of `rcu_interrupt_disable` is shown as below:

Table 3-606. Function `rcu_interrupt_disable`

Function name	rcu_interrupt_disable
Function prototype	void rcu_interrupt_disable(rcu_int_enum stab_int);
Function descriptions	disable the stabilization interrupt
Precondition	-
The called functions	-
Input parameter{in}	
stab_int	clock stabilization interrupt, refer to Table 3-573. Enum rcu_int_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

rcu_lxtal_drive_capability_config

The description of `rcu_lxtal_drive_capability_config` is shown as below:

Table 3-607. Function `rcu_lxtal_drive_capability_config`

Function name	<code>rcu_lxtal_drive_capability_config</code>
Function prototype	<code>void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);</code>
Function descriptions	configure the LXTAL drive capability
Precondition	-
The called functions	-
Input parameter{in}	
<code>lxtal_dricap</code>	drive capability of LXTAL
<code>RCU_LXTAL_LOWDRI</code>	lower driving capability
<code>RCU_LXTAL_HIGHDRI</code>	higher driving capability
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the LXTAL drive capability */
rcu_lxtal_drive_capability_config(RCU_LXTAL_LOWDRI);
```

rcu_osc_stab_wait

The description of `rcu_osc_stab_wait` is shown as below:

Table 3-608. Function `rcu_osc_stab_wait`

Function name	<code>rcu_osc_stab_wait</code>
Function prototype	<code>ErrStatus rcu_osc_stab_wait(rcu_osc_type_enum osci);</code>
Function descriptions	wait for oscillator stabilization flags is SET or oscillator startup is timeout
Precondition	-
The called functions	<code>rcu_flag_get</code>
Input parameter{in}	

osci	oscillator types, refer to Table 3-574. Enum rcu_osc_type enum
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* wait for oscillator stabilization flag */

if(SUCCESS == rcu_osc_stab_wait(RCU_HXTAL)){
}
```

rcu_osc_on

The description of `rcu_osc_on` is shown as below:

Table 3-609. Function `rcu_osc_on`

Function name	rcu_osc_on
Function prototype	void rcu_osc_on(rcu_osc_type_enum osci);
Function descriptions	turn on the oscillator
Precondition	-
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to Table 3-574. Enum rcu_osc_type enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn on the high speed crystal oscillator */

rcu_osc_on(RCU_HXTAL);
```

rcu_osc_off

The description of `rcu_osc_off` is shown as below:

Table 3-610. Function rcu_osci_off

Function name	rcu_osci_off
Function prototype	void rcu_osci_off(rcu_osci_type_enum osci);
Function descriptions	turn off the oscillator
Precondition	-
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to Table 3-574. Enum rcu_osci_type_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn off the high speed crystal oscillator */
rcu_osci_off(RCU_HXTAL);
```

rcu_osci_bypass_mode_enable

The description of rcu_osci_bypass_mode_enable is shown as below:

Table 3-611. Function rcu_osci_bypass_mode_enable

Function name	rcu_osci_bypass_mode_enable
Function prototype	void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);
Function descriptions	enable the oscillator bypass mode
Precondition	HXTALEN or LXTALEN must be reset before it
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to Table 3-574. Enum rcu_osci_type_enum
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable the high speed crystal oscillator bypass mode */
rcu_osc_bypass_mode_enable(RCU_HXTAL);
```

rcu_osc_bypass_mode_disable

The description of `rcu_osc_bypass_mode_disable` is shown as below:

Table 3-612. Function `rcu_osc_bypass_mode_disable`

Function name	rcu_osc_bypass_mode_disable
Function prototype	void rcu_osc_bypass_mode_disable(rcu_osc_type_enum osci);
Function descriptions	disable the oscillator bypass mode
Precondition	HXTALEN or LXTALEN must be reset before it
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to Table 3-574. Enum <code>rcu_osc_type_enum</code>
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the high speed crystal oscillator bypass mode */
rcu_osc_bypass_mode_disable(RCU_HXTAL);
```

rcu_hxtal_clock_monitor_enable

The description of `rcu_hxtal_clock_monitor_enable` is shown as below:

Table 3-613. Function rcu_hxtal_clock_monitor_enable

Function name	rcu_hxtal_clock_monitor_enable
Function prototype	void rcu_hxtal_clock_monitor_enable(void);
Function descriptions	enable the HXTAL clock monitor
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the HXTAL clock monitor */

rcu_hxtal_clock_monitor_enable();
```

rcu_hxtal_clock_monitor_disable

The description of rcu_hxtal_clock_monitor_disable is shown as below:

Table 3-614. Function rcu_hxtal_clock_monitor_disable

Function name	rcu_hxtal_clock_monitor_disable
Function prototype	void rcu_hxtal_clock_monitor_disable(void);
Function descriptions	disable the HXTAL clock monitor
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable the HXTAL clock monitor */

rcu_hxtal_clock_monitor_disable();
```

rcu_irc16m_adjust_value_set

The description of `rcu_irc16m_adjust_value_set` is shown as below:

Table 3-615. Function `rcu_irc16m_adjust_value_set`

Function name	rcu_irc16m_adjust_value_set
Function prototype	void rcu_irc16m_adjust_value_set(uint8_t irc16m_adjval);
Function descriptions	set the IRC16M adjust value
Precondition	-
The called functions	-
Input parameter{in}	
Irc16m_adjval	IRC16M adjust value, must be between 0 and 0x1F
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the IRC16M adjust value */

rcu_irc16m_adjust_value_set(0x10);
```

rcu_spread_spectrum_config

The description of `rcu_spread_spectrum_config` is shown as below:

Table 3-616. Function `rcu_spread_spectrum_config`

Function name	rcu_spread_spectrum_config
Function prototype	void rcu_spread_spectrum_config(uint32_t spread_spectrum_type, uint32_t modstep, uint32_t modcnt)
Function descriptions	configure the spread spectrum modulation for the main PLL clock

Precondition	-
The called functions	-
Input parameter{in}	
spread_spectrum_type	PLL spread spectrum modulation type select
<i>RCU_SS_TYPE_CENTER</i>	center spread type is selected
<i>RCU_SS_TYPE_DOWN</i>	down spread type is selected
Input parameter{in}	
modstep	configure PLL spread spectrum modulation profile amplitude
<i>uint32_t</i>	0 ~ 0x7FFF, The following criteria must be met: MODSTEP*MODCNT $\leq 2^{15-1}$
Input parameter{in}	
modcnt	configure PLL spread spectrum modulation profile frequency
<i>uint32_t</i>	0 ~ 0x1FFF, The following criteria must be met: MODSTEP*MODCNT $\leq 2^{15-1}$
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure PLL spread_spectrum */
rcu_spread_spectrum_config (RCU_SS_TYPE_CENTER, 0x0F,0x0F);
```

rcu_spread_spectrum_enable

The description of `rcu_spread_spectrum_enable` is shown as below:

Table 3-617. Function `rcu_spread_spectrum_enable`

Function name	<code>rcu_spread_spectrum_enable</code>
Function prototype	<code>void rcu_spread_spectrum_enable(void);</code>
Function descriptions	enable the PLL spread spectrum modulation

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the PLL spread spectrum modulation */
rcu_spread_spectrum_enable();
```

rcu_spread_spectrum_disable

The description of `rcu_spread_spectrum_disable` is shown as below:

Table 3-618. Function `rcu_spread_spectrum_disable`

Function name	rcu_spread_spectrum_disable
Function prototype	void rcu_spread_spectrum_disable(void);
Function descriptions	disable the HXTAL clock monitor
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the PLL spread spectrum modulation */
rcu_spread_spectrum_disable();
```

rcu_voltage_key_unlock

The description of `rcu_voltage_key_unlock` is shown as below:

Table 3-619. Function `rcu_voltage_key_unlock`

Function name	rcu_voltage_key_unlock
Function prototype	void <code>rcu_voltage_key_unlock</code> (void);
Function descriptions	unlock the voltage key
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock the voltage key register */
rcu_voltage_key_unlock();
```

rcu_deepsleep_voltage_set

The description of `rcu_deepsleep_voltage_set` is shown as below:

Table 3-620. Function `rcu_deepsleep_voltage_set`

Function name	rcu_deepsleep_voltage_set
Function prototype	void <code>rcu_deepsleep_voltage_set</code> (uint32_t dsvol);
Function descriptions	set the deep-sleep mode voltage value
Precondition	-
The called functions	-
Input parameter{in}	
dsvol	deep sleep mode voltage
RCU_DEEPSLEEP_V_	the core voltage is default value in deep-sleep mode

<i>O</i>	
<i>RCU_DEEPSLEEP_V_1</i>	the core voltage is (default value-0.1)V in deep-sleep mode(customers are not recommended to use it)
<i>RCU_DEEPSLEEP_V_2</i>	the core voltage is (default value-0.2)V in deep-sleep mode(customers are not recommended to use it)
<i>RCU_DEEPSLEEP_V_3</i>	the core voltage is (default value-0.3)V in deep-sleep mode(customers are not recommended to use it)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the deep-sleep mode voltage */
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1);
```

rcu_clock_freq_get

The description of `rcu_clock_freq_get` is shown as below:

Table 3-621. Function `rcu_clock_freq_get`

Function name	rcu_clock_freq_get
Function prototype	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
Function descriptions	get the system clock, bus clock frequency
Precondition	-
The called functions	-
Input parameter{in}	
<i>clock</i>	the clock frequency which to get, refer to Table 3-575. Enum rcu_clock_freq_enum
Output parameter{out}	
-	-
Return value	
<i>ck_freq</i>	clock frequency of system, AHB, APB1, APB2

Example:

```

uint32_t temp_freq;

/* get the system clock frequency */

temp_freq = rcu_clock_freq_get(CK_SYS);

```

3.22. RTC

The Real-time Clock (RTC) is usually used as a clock-calendar. The ones in the Backup Domain consist of a 32-bit up-counter, an alarm, a prescaler, a divider and the RTC clock configuration register. The RTC registers are listed in chapter [3.22.1](#), the RTC firmware functions are introduced in chapter[3.22.2](#).

3.22.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

Table 3-622. RTC Registers

Registers	Descriptions
RTC_TIME	RTC time of day register
RTC_DATE	RTC date register
RTC_CTL	RTC control register
RTC_STAT	RTC status register
RTC_PSC	RTC time prescaler register
RTC_WUT	RTC wakeup timer register
RTC_COSC	RTC coarse calibration register
RTC_ALRM0TD	RTC alarm 0 time and date register
RTC_ALRM1TD	RTC alarm 1 time and date register
RTC_WPK	RTC write protection key register
RTC_SS	RTC sub second register
RTC_SHIFTCTL	RTC shift function control register
RTC_TTS	RTC time of timestamp register
RTC_DTS	RTC date of timestamp register
RTC_SSTS	RTC sub second of timestamp register
RTC_HRFC	RTC high resolution frequency compensation register

Registers	Descriptions
RTC_TAMP	RTC tamper register
RTC_ALRM0SS	RTC alarm 0 sub second register
RTC_ALRM1SS	RTC alarm 1 sub second register
RTC_BKPx(x = 0, 1, 2, ..., 18, 19)	RTC backup register

3.22.2. Descriptions of Peripheral functions

RTC firmware functions are listed in the table shown as below:

Table 3-623.RTC firmware function

Function name	Function description
rtc_deinit	reset most of the RTC registers
rtc_init	initialize RTC registers
rtc_init_mode_enter	enter RTC init mode
rtc_init_mode_exit	exit RTC init mode
rtc_register_sync_wait	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
rtc_current_time_get	get current time and date
rtc_subsecond_get	get current subsecond value
rtc_alarm_config	configure RTC alarm
rtc_alarm_subsecond_config	configure subsecond of RTC alarm
rtc_alarm_get	get RTC alarm
rtc_alarm_subsecond_get	get RTC alarm subsecond
rtc_alarm_enable	enable RTC alarm
rtc_alarm_disable	disable RTC alarm
rtc_timestamp_enable	enable RTC time-stamp
rtc_timestamp_disable	disable RTC time-stamp
rtc_timestamp_get	get RTC timestamp time and date
rtc_timestamp_subsecond_get	get RTC time-stamp subsecond

Function name	Function description
rtc_timestamp_pin_map	RTC time-stamp pin map
rtc_tamper_enable	enable RTC tamper
rtc_tamper_disable	disable RTC tamper
rtc_tamper0_pin_map	RTC tamper0 pin map
rtc_interrupt_enable	enable specified RTC interrupt
rtc_interrupt_disable	disble specified RTC interrupt
rtc_flag_get	check specified flag
rtc_flag_clear	clear specified flag
rtc_alter_output_config	configure RTC alternate output source
rtc_calibration_output_config	configure RTC calibration output source
rtc_hour_adjust	ajust the daylight saving time by adding or substracting one hour from the current time
rtc_second_adjust	ajust RTC second or subsecond value of current time
rtc_bypass_shadow_enable	enable RTC bypass shadow registers function
rtc_bypass_shadow_disable	disable RTC bypass shadow registers function
rtc_refclock_detection_enable	enable RTC reference clock detection function
rtc_refclock_detection_disable	disable RTC reference clock detection function
rtc_wakeup_enable	enable RTC wakeup timer
rtc_wakeup_disable	disable RTC wakeup timer
rtc_wakeup_clock_set	set auto wakeup timer clock
rtc_wakeup_timer_set	set auto wakeup timer value
rtc_wakeup_timer_get	get auto wakeup timer value
rtc_smooth_calibration_config	configure RTC smooth calibration
rtc_coarse_calibration_enable	enable RTC coarse calibration
rtc_coarse_calibration_disable	disable RTC coarse calibration
rtc_coarse_calibration_config	configure RTC coarse calibration direction and step

Structure rtc_parameter_struct

Table 3-624. Struct rtc_parameter_struct

Member name	Function description
year	RTC year value: 0x0 - 0x99(BCD format)
month	RTC month value
date	RTC date value: 0x1 - 0x31(BCD format)
day_of_week	RTC weekday value
hour	RTC hour value
minute	RTC minute value: 0x0 - 0x59(BCD format)
second	RTC second value: 0x0 - 0x59(BCD format)
factor_asyn	RTC asynchronous prescaler value: 0x0 - 0x7F
factor_syn	RTC synchronous prescaler value: 0x0 - 0xFFFF
am_pm	RTC AM/PM value
display_format	RTC time notation

Structure rtc_alarm_struct

Table 3-625. Struct rtc_alarm_struct

Member name	Function description
alarm_mask	RTC alarm mask
weekday_or_date	specify RTC alarm is on date or weekday
alarm_day	RTC alarm date or weekday value
alarm_hour	RTC alarm hour value
alarm_minute	RTC alarm minute value: 0x0 - 0x59(BCD format)
alarm_second	RTC alarm second value: 0x0 - 0x59(BCD format)
am_pm	RTC alarm AM/PM value

Structure rtc_timestamp_struct

Table 3-626. Struct rtc_timestamp_struct

Member name	Function description
timestamp_month	RTC time-stamp month value

timestamp_date	RTC time-stamp date value: 0x1 - 0x31(BCD format)
timestamp_day	RTC time-stamp weekday value
timestamp_hour	RTC time-stamp hour value
timestamp_minute	RTC time-stamp minute value: 0x0 - 0x59(BCD format)
timestamp_second	RTC time-stamp second value: 0x0 - 0x59(BCD format)
am_pm	RTC time-stamp AM/PM value

Structure `rtc_tamper_struct`

Table 3-627. Struct `rtc_tamper_struct`

Member name	Function description
tamper_source	RTC tamper source
tamper_trigger	RTC tamper trigger
tamper_filter	RTC tamper consecutive samples needed during a voltage level detection
tamper_sample_frequency	RTC tamper sampling frequency during a voltage level detection
tamper_precharge_enable	RTC tamper precharge feature during a voltage level detection
tamper_precharge_time	RTC tamper precharge duration if precharge feature is enabled
tamper_with_timestamp	RTC tamper time-stamp feature

`rtc_deinit`

The description of `rtc_deinit` is shown as below:

Table 3-628. Function `rtc_deinit`

Function name	rtc_deinit
Function prototype	ErrStatus rtc_deinit(void);
Function descriptions	reset most of the RTC registers
Precondition	-
The called functions	rcu_periph_reset_enable/ rcu_periph_reset_disable -
Input parameter{in}	

-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* reset most of the RTC registers*/
```

```
ErrStatus error_status = rtc_deinit();
```

rtc_init

The description of `rtc_init` is shown as below:

Table 3-629. Function `rtc_init`

Function name	rtc_init
Function prototype	ErrStatus rtc_init(rtc_parameter_struct* rtc_initpara_struct);
Function descriptions	initialize RTC registers
Precondition	-
The called functions	-
Input parameter{in}	
rtc_initpara_struct	the structure members can refer to members of the structure Table 3-624. <u>Struct rtc_parameter_struct</u>
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* reset most of the RTC registers*/
```

```
rtc_parameter_struct rtc_initpara_struct;
```

```
ErrStatus error_status = rtc_init (&rtc_initpara_struct);
```

rtc_init_mode_enter

The description of `rtc_init_mode_enter` is shown as below:

Table 3-630. Function `rtc_init_mode_enter`

Function name	rtc_init_mode_enter
Function prototype	ErrStatus rtc_init_mode_enter(void);
Function descriptions	enter RTC init mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/*enter RTC init mode*/
ErrStatus error_status = rtc_init_mode_enter();
```

rtc_init_mode_exit

The description of `rtc_init_mode_exit` is shown as below:

Table 3-631. Function `rtc_init_mode_exit`

Function name	rtc_init_mode_exit
Function prototype	void rtc_init_mode_exit(void);
Function descriptions	exit RTC init mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/*exit RTC init mode*/
rtc_init_mode_exit();
```

rtc_register_sync_wait

The description of `rtc_register_sync_wait` is shown as below:

Table 3-632. Function `rtc_register_sync_wait`

Function name	rtc_register_sync_wait
Function prototype	ErrStatus rtc_register_sync_wait(void);
Function descriptions	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/*wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the
shadow registers are updated*/
```

```
ErrStatus error_status = rtc_register_sync_wait();
```

rtc_current_time_get

The description of `rtc_current_time_get` is shown as below:

Table 3-633. Function rtc_current_time_get

Function name	rtc_current_time_get
Function prototype	void rtc_current_time_get(rtc_parameter_struct* rtc_initpara_struct);
Function descriptions	get current time and date
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
rtc_initpara_struct	the structure members can refer to members of the structure Table 3-624. <u>Struct rtc_parameter_struct</u>
Return value	
-	-

Example:

```
/*get current time and date*/
rtc_parameter_struct rtc_initpara_struct;
rtc_current_time_get (&rtc_initpara_struct);
```

rtc_subsecond_get

The description of rtc_subsecond_get is shown as below:

Table 3-634. Function rtc_subsecond_get

Function name	rtc_subsecond_get
Function prototype	uint32_t rtc_subsecond_get(void);
Function descriptions	get current subsecond value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
<code>uint32_t</code>	current subsecond value(0x00-0xFFFF)

Example:

```
/*get current subsecond value*/
uint32_t sub_second = rtc_subsecond_get();
```

rtc_alarm_config

The description of `rtc_alarm_config` is shown as below:

Table 3-635. Function `rtc_alarm_config`

Function name	<code>rtc_alarm_config</code>
Function prototype	<code>void rtc_alarm_config(uint8_t rtc_alarm, rtc_alarm_struct* rtc_alarm_time);</code>
Function descriptions	configure RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
<code>rtc_alarm</code>	select alarm0 or alarm1
<code>RTC_ALARM0</code>	alarm0
<code>RTC_ALARM1</code>	alarm1
Input parameter{in}	
<code>rtc_alarm_time</code>	the structure members can refer to members of the structure Table 3-625. <u>Struct rtc_alarm_struct</u>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure RTC alarm0*/
rtc_alarm_struct rtc_alarm_time;
rtc_alarm_config(RTC_ALARM0, &rtc_alarm_time);
```

rtc_alarm_subsecond_config

The description of `rtc_alarm_subsecond_config` is shown as below:

Table 3-636. Function `rtc_alarm_subsecond_config`

Function name	<code>rtc_alarm_subsecond_config</code>
Function prototype	<code>void rtc_alarm_subsecond_config(uint8_t rtc_alarm, uint32_t mask_subsecond, uint32_t subsecond);</code>
Function descriptions	configure subsecond of RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
<code>rtc_alarm</code>	select alarm0 or alarm1
<code>RTC_ALARM0</code>	alarm0
<code>RTC_ALARM1</code>	alarm1
Input parameter{in}	
<code>mask_subsecond</code>	alarm subsecond mask
<code>RTC_MASKSSC_0_14</code>	mask alarm subsecond configuration
<code>RTC_MASKSSC_1_14</code>	mask RTC_ALRM0SS_SSC[14:1], and RTC_ALRM0SS_SSC[0] is to be compared
<code>RTC_MASKSSC_2_14</code>	mask RTC_ALRM0SS_SSC[14:2], and RTC_ALRM0SS_SSC[1:0] is to be compared
<code>RTC_MASKSSC_3_14</code>	mask RTC_ALRM0SS_SSC[14:3], and RTC_ALRM0SS_SSC[2:0] is to be compared
<code>RTC_MASKSSC_4_14</code>	mask RTC_ALRM0SS_SSC[14:4], and RTC_ALRM0SS_SSC[3:0] is to be compared
<code>RTC_MASKSSC_5_14</code>	mask RTC_ALRM0SS_SSC[14:5], and RTC_ALRM0SS_SSC[4:0] is to be compared
<code>RTC_MASKSSC_6_14</code>	mask RTC_ALRM0SS_SSC[14:6], and RTC_ALRM0SS_SSC[5:0] is to be compared
<code>RTC_MASKSSC_7_14</code>	mask RTC_ALRM0SS_SSC[14:7], and RTC_ALRM0SS_SSC[6:0] is to be compared
<code>RTC_MASKSSC_8_14</code>	mask RTC_ALRM0SS_SSC[14:8], and RTC_ALRM0SS_SSC[7:0] is to be compared

	compared
<i>RTC_MASKSSC_9_14</i>	mask RTC_ALRM0SS_SSC[14:9], and RTC_ALRM0SS_SSC[8:0] is to be compared
<i>RTC_MASKSSC_10_14</i>	mask RTC_ALRM0SS_SSC[14:10], and RTC_ALRM0SS_SSC[9:0] is to be compared
<i>RTC_MASKSSC_11_14</i>	mask RTC_ALRM0SS_SSC[14:11], and RTC_ALRM0SS_SSC[10:0] is to be compared
<i>RTC_MASKSSC_12_14</i>	mask RTC_ALRM0SS_SSC[14:12], and RTC_ALRM0SS_SSC[11:0] is to be compared
<i>RTC_MASKSSC_13_14</i>	mask RTC_ALRM0SS_SSC[14:13], and RTC_ALRM0SS_SSC[12:0] is to be compared
<i>RTC_MASKSSC_14</i>	mask RTC_ALRM0SS_SSC[14], and RTC_ALRM0SS_SSC[13:0] is to be compared
<i>RTC_MASKSSC_NONE</i>	mask none, and RTC_ALRM0SS_SSC[14:0] is to be compared
Input parameter{in}	
subsecond	alarm subsecond value(0x000 - 0x7FFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*configure subsecond of RTC alarm0*/
rtc_alarm_subsecond_config (RTC_ALARM0, RTC_MASKSSC_9_14, 0x7FFF);

rtc_alarm_get
```

The description of **rtc_alarm_get** is shown as below:

Table 3-637. Function **rtc_alarm_get**

Function name	rtc_alarm_get
Function prototype	void rtc_alarm_get(uint8_t rtc_alarm, rtc_alarm_struct* rtc_alarm_time);
Function descriptions	get RTC alarm
Precondition	-

The called functions	-
Input parameter{in}	
rtc_alarm	select alarm0 or alarm1
<i>RTC_ALARM0</i>	alarm0
<i>RTC_ALARM1</i>	alarm1
Output parameter{out}	
rtc_alarm_time	the structure members can refer to members of the structure Table 3-625. <u>Struct rtc_alarm_struct</u>
Return value	
-	-

Example:

```
/*get RTC alarm0*/
rtc_alarm_struct rtc_alarm_time;
rtc_alarm_get (RTC_ALARM0, &rtc_alarm_time);
```

rtc_alarm_subsecond_get

The description of `rtc_alarm_subsecond_get` is shown as below:

Table 3-638. Function `rtc_alarm_subsecond_get`

Function name	rtc_alarm_subsecond_get
Function prototype	uint32_t rtc_alarm_subsecond_get(uint8_t rtc_alarm);
Function descriptions	get RTC alarm subsecond
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm0 or alarm1
<i>RTC_ALARM0</i>	alarm0
<i>RTC_ALARM1</i>	alarm1
Output parameter{out}	
-	-

Return value	
<code>uint32_t</code>	RTC alarm subsecond value(0x0-0x3FFF)

Example:

```
/*get RTC alarm0 subsecond*/
uint32_t subsecond = rtc_alarm_subsecond_get(RTC_ALARM0);
```

rtc_alarm_enable

The description of `rtc_alarm_enable` is shown as below:

Table 3-639. Function `rtc_alarm_enable`

Function name	rtc_alarm_enable
Function prototype	<code>void rtc_alarm_enable(uint8_t rtc_alarm);</code>
Function descriptions	enable RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
<code>rtc_alarm</code>	select alarm0 or alarm1
<code>RTC_ALARM0</code>	alarm0
<code>RTC_ALARM1</code>	alarm1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*enable RTC alarm0*/
rtc_alarm_enable(RTC_ALARM0);
```

rtc_alarm_disable

The description of `rtc_alarm_disable` is shown as below:

Table 3-640. Function `rtc_alarm_disable`

Function name	rtc_alarm_disable
----------------------	-------------------

Function prototype	ErrStatus rtc_alarm_disable(uint8_t rtc_alarm);
Function descriptions	disable RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm0 or alarm1
RTC_ALARM0	alarm0
RTC_ALARM1	alarm1
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/*disable RTC alarm0*/
ErrStatus error_status = rtc_alarm_disable(RTC_ALARM0);
```

rtc_timestamp_enable

The description of rtc_timestamp_enable is shown as below:

Table 3-641. Function rtc_timestamp_enable

Function name	rtc_timestamp_enable
Function prototype	void rtc_timestamp_enable(uint32_t edge);
Function descriptions	enable RTC time-stamp
Precondition	-
The called functions	-
Input parameter{in}	
edge	specify which edge to detect of time-stamp
RTC_TIMESTAMP_RISING_EDGE	rising edge is valid event edge for timestamp event
RTC_TIMESTAMP_FALLING_EDGE	falling edge is valid event edge for timestamp event

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*enable RTC time-stamp*/
rtc_timestamp_enable (RTC_TIMESTAMP_RISING_EDGE);
```

rtc_timestamp_disable

The description of `rtc_timestamp_disable` is shown as below:

Table 3-642. Function `rtc_timestamp_disable`

Function name	rtc_timestamp_disable
Function prototype	void rtc_timestamp_disable(void);
Function descriptions	disable RTC time-stamp
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*disable RTC time-stamp*/
rtc_timestamp_disable ();
```

rtc_timestamp_get

The description of `rtc_timestamp_get` is shown as below:

Table 3-643. Function `rtc_timestamp_get`

Function name	rtc_timestamp_get
----------------------	-------------------

Function prototype	void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);
Function descriptions	get RTC timestamp time and date
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
rtc_timestamp	the structure members can refer to members of the structure Table 3-627. <u>Struct rtc_tamper struct</u>
Return value	
-	-

Example:

```

/* get RTC timestamp time and date */

rtc_timestamp_struct rtc_timestamp;

rtc_timestamp_get(& rtc_timestamp);

```

rtc_timestamp_subsecond_get

The description of `rtc_timestamp_subsecond_get` is shown as below:

Table 3-644. Function `rtc_timestamp_subsecond_get`

Function name	rtc_timestamp_subsecond_get
Function prototype	uint32_t rtc_timestamp_subsecond_get(void);
Function descriptions	get RTC time-stamp subsecond
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

uint32_t	RTC time-stamp subsecond value
-----------------	--------------------------------

Example:

```
/* get RTC time-stamp subsecond */

uint32_t subsecond = rtc_timestamp_subsecond_get();
```

rtc_timestamp_pin_map

The description of **rtc_timestamp_pin_map** is shown as below:

Table 3-645. Function `rtc_timestamp_pin_map`

Function name	rtc_timestamp_pin_map
Function prototype	void rtc_timestamp_pin_map(uint32_t rtc_af);
Function descriptions	RTC time-stamp mapping
Precondition	-
The called functions	-
Input parameter{in}	
rtc_af	Timestamp input mapping selection
<i>RTC_AF0_TIMESTAMP</i> <i>P</i>	RTC_AF0 use for timestamp
<i>RTC_AF1_TIMESTAMP</i> <i>P</i>	RTC_AF1 use for timestamp
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* RTC time-stamp mapping */

rtc_timestamp_pin_map (RTC_AF0_TIMESTAMP);
```

rtc_tamper_enable

The description of **rtc_tamper_enable** is shown as below:

Table 3-646. Function rtc_timestamp_enable

Function name	rtc_tamper_enable
Function prototype	void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);
Function descriptions	enable RTC tamper
Precondition	-
The called functions	-
Input parameter{in}	
rtc_tamper	the structure members can refer to members of the structure Table 3-627. <u>Struct rtc_tamper_struct</u>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable RTC tamper */

rtc_tamper_struct rtc_tamper

rtc_tamper_enable(& rtc_tamper);
```

rtc_tamper_disable

The description of rtc_tamper_disable is shown as below:

Table 3-647. Function rtc_tamper_disable

Function name	rtc_tamper_disable
Function prototype	void rtc_tamper_disable(uint32_t source);
Function descriptions	disable RTC tamper
Precondition	-
The called functions	-
Input parameter{in}	
source	specify which tamper source to be disabled
RTC_TAMPER0	RTC tamper0
RTC_TAMPER1	RTC tamper1

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RTC tamper */
rtc_tamper_disable(RTC_TAMPER0);
```

rtc_tamper0_pin_map

The description of rtc_tamper0_pin_map is shown as below:

Table 3-648. Function rtc_tamper0_pin_map

Function name	rtc_tamper0_pin_map
Function prototype	void rtc_tamper0_pin_map (uint32_t rtc_af);
Function descriptions	RTC tamper0 mapping
Precondition	-
The called functions	-
Input parameter{in}	
rtc_af	Tamper0 fuction input mapping selection
RTC_AF0_TAMPER0	RTC_AF0 use for tamper0
RTC_AF1_TAMPER0	RTC_AF1 use for tamper0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* RTC tamper0 mapping */
rtc_tamper0_pin_map (RTC_AF0_TAMPER0);
```

rtc_interrupt_enable

The description of rtc_interrupt_enable is shown as below:

Table 3-649. Function rtc_interrupt_enable

Function name	rtc_interrupt_enable
Function prototype	void rtc_interrupt_enable(uint32_t interrupt);
Function descriptions	enable specified RTC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify which interrupt source to be enabled
<i>RTC_INT_TIMESTAMP</i>	timestamp interrupt
<i>RTC_INT_ALARM0</i>	Alarm0 interrupt
<i>RTC_INT_ALARM1</i>	Alarm1 interrupt
<i>RTC_INT_TAMP</i>	tamp interrupt
<i>RTC_INT_WAKEUP</i>	wakeup timer interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable specified RTC interrupt*/
rtc_interrupt_enable(RTC_INT_TAMP);
```

rtc_interrupt_disable

The description of **rtc_interrupt_disable** is shown as below:

Table 3-650. Function rtc_interrupt_disable

Function name	rtc_interrupt_disable
Function prototype	void rtc_interrupt_disable(uint32_t interrupt);
Function descriptions	disable specified RTC interrupt
Precondition	-
The called functions	-

Input parameter{in}	
interrupt	specify which RTC interrupt to disable
<i>RTC_INT_TIMESTAMP</i>	second interrupt
<i>RTC_INT_ALARM0</i>	alarm0 interrupt
<i>RTC_INT_ALARM1</i>	alarm1 interrupt
<i>RTC_INT_TAMP</i>	tamp interrupt
<i>RTC_INT_WAKEUP</i>	wakeup timer interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable specified RTC interrupt */
rtc_interrupt_disable(RTC_INT_TAMP);
```

rtc_flag_get

The description of **rtc_flag_get** is shown as below:

Table 3-651. Function rtc_flag_get

Function name	rtc_flag_get
Function prototype	FlagStatus rtc_flag_get(uint32_t flag);
Function descriptions	check specified flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	specify which flag to check
<i>RTC_STAT_SCP</i>	smooth calibration pending flag
<i>RTC_FLAG_TP1</i>	RTC tamper 1 detected flag
<i>RTC_FLAG_TP0</i>	RTC tamper 0 detected flag
<i>RTC_FLAG_TSOVR</i>	time-stamp overflow flag

<i>RTC_FLAG_TS</i>	time-stamp flag
<i>RTC_FLAG_ALARM0</i>	alarm0 occurs flag
<i>RTC_FLAG_ALARM1</i>	Alarm1 occurs flag
<i>RTC_FLAG_WT</i>	wakeup timer occurs flag
<i>RTC_FLAG_INIT</i>	initialization state flag
<i>RTC_FLAG_RSYN</i>	register synchronization flag
<i>RTC_FLAG_YCM</i>	year configuration mark status flag
<i>RTC_FLAG_SOP</i>	alarm0 configuration can be write flag
<i>RTC_FLAG_ALRM0W</i>	alarm0 writen available flag
<i>RTC_FLAG_ALRM1W</i>	alarm1 writen available flag
<i>RTC_FLAG_WTW</i>	wakeup timer can be write flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check time-stamp event flag */
FlagStatus = rtc_flag_get(RTC_FLAG_TIMESTAMP)
```

rtc_flag_clear

The description of `rtc_flag_clear` is shown as below:

Table 3-652. Function `rtc_flag_clear`

Function name	<code>rtc_flag_clear</code>
Function prototype	<code>void rtc_flag_clear(uint32_t flag);</code>
Function descriptions	clear specified flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	specify which flag to clear

<i>RTC_FLAG_TP1</i>	RTC tamper 1 detected flag
<i>RTC_FLAG_TP0</i>	RTC tamper 0 detected flag
<i>RTC_FLAG_TSOVR</i>	time-stamp overflow flag
<i>RTC_FLAG_TS</i>	time-stamp flag
<i>RTC_FLAG_WT</i>	wakeup timer occurs flag
<i>RTC_FLAG_ALARM0</i>	alarm0 occurs flag
<i>RTC_FLAG_ALARM1</i>	alarm1 occurs flag
<i>RTC_FLAG_RSYN</i>	register synchronization flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* cleartime-stamp event flag */
rtc_flag_clear (RTC_FLAG_TIMESTAMP);
```

rtc_alter_output_config

The description of `rtc_alter_output_config` is shown as below:

Table 3-653. Function `rtc_alter_output_config`

Function name	<code>rtc_alter_output_config</code>
Function prototype	<code>void rtc_alter_output_config(uint32_t source, uint32_t mode);</code>
Function descriptions	configure rtc alternate output source
Precondition	-
The called functions	-
Input parameter{in}	
source	specify signal to output
<i>RTC_ALARM0_HIGH</i>	when the alarm0 flag is set, the output pin is high
<i>RTC_ALARM0_LOW</i>	when the alarm0 flag is set, the output pin is low
<i>RTC_ALARM1_HIGH</i>	when the alarm1 flag is set, the output pin is high

<i>RTC_ALARM1_LOW</i>	when the alarm1 flag is set, the output pin is low
<i>RTC_WAKEUP_HIGH</i>	when the wakeup flag is set, the output pin is high
<i>RTC_WAKEUP_LOW</i>	when the wakeup flag is set, the output pin is low
Input parameter{in}	
mode	specify the output pin (PC13) mode when output alarm signal
<i>RTC_ALARM_OUTPUT_OD</i>	open drain mode
<i>RTC_ALARM_OUTPUT_PP</i>	push pull mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure rtc alternate output source */
rtc_alter_output_config(RTC_ALARM0_LOW, RTC_ALARM_OUTPUT_PP);
```

rtc_calibration_output_config

The description of `rtc_calibration_output_config` is shown as below:

Table 3-654. Function `rtc_calibration_output_config`

Function name	rtc_calibration_output_config
Function prototype	void rtc_calibration_output_config(uint32_t source)
Function descriptions	configure rtc calibration output source
Precondition	-
The called functions	-
Input parameter{in}	
source	specify signal to output
<i>RTC_CALIBRATION_5</i> <i>12HZ</i>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 512Hz signal
<i>RTC_CALIBRATION_1</i>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value,

HZ	output 1Hz signal
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure rtc calibration output source*/
rtc_calibration_output_config (RTC_CALIBRATION_1HZ);
```

rtc_hour_adjust

The description of `rtc_hour_adjust` is shown as below:

Table 3-655. Function `rtc_hour_adjust`

Function name	rtc_hour_adjust
Function prototype	void rtc_hour_adjust(uint32_t operation);
Function descriptions	adjust the daylight saving time by adding or subtracting one hour from the current time
Precondition	-
The called functions	-
Input parameter{in}	
operation	hour adjustment operation
<i>RTC_CTL_A1H</i>	add one hour
<i>RTC_CTL_S1H</i>	subtract one hour
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* adjust the daylight saving time by adding one hour from the current time */
rtc_hour_adjust(RTC_CTL_A1H);
```

rtc_second_adjust

The description of `rtc_second_adjust` is shown as below:

Table 3-656. Function `rtc_second_adjust`

Function name	rtc_second_adjust
Function prototype	ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);
Function descriptions	adjust RTC second or subsecond value of current time
Precondition	-
The called functions	-
Input parameter{in}	
add	add 1s to current time or not
<i>RTC_SHIFT_ADD1S_R ESET</i>	no effect
<i>RTC_SHIFT_ADD1S_S ET</i>	add 1s to current time
Input parameter{in}	
minus	number of subsecond to minus from current time(0x0 - 0x7FFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* adjust RTC second or subsecond value of current time */

ErrStatus error_status = rtc_second_adjust(RTC_SHIFT_ADD1S_SET, 0);
```

rtc_bypass_shadow_enable

The description of `rtc_bypass_shadow_enable` is shown as below:

Table 3-657. Function `rtc_bypass_shadow_enable`

Function name	rtc_bypass_shadow_enable
Function prototype	void rtc_bypass_shadow_enable(void);
Function descriptions	enable RTC bypass shadow registers function

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable RTC bypass shadow registers function*/
rtc_bypass_shadow_enable();
```

rtc_bypass_shadow_disable

The description of `rtc_bypass_shadow_disable` shown as below:

Table 3-658. Function `rtc_bypass_shadow_disable`

Function name	rtc_bypass_shadow_disable
Function prototype	void rtc_bypass_shadow_disable (void);
Function descriptions	disable RTC bypass shadow registers function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RTC bypass shadow registers function*/
rtc_bypass_shadow_disable();
```

rtc_refclock_detection_enable

The description of `rtc_refclock_detection_enable` shown as below:

Table 3-659. Function `rtc_refclock_detection_enable`

Function name	rtc_refclock_detection_enable
Function prototype	ErrStatus rtc_refclock_detection_enable(void);
Function descriptions	enable RTC reference clock detection function
Precondition	-
The called functions	rtc_init_mode_enter/rtc_init_mode_exit
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* enable RTC reference clock detection function*/
ErrStatus error_status = rtc_refclock_detection_enable();
```

rtc_refclock_detection_disable

The description of `rtc_refclock_detection_disable` shown as below:

Table 3-660. Function `rtc_refclock_detection_disable`

Function name	rtc_refclock_detection_disable
Function prototype	ErrStatus rtc_refclock_detection_disable(void);
Function descriptions	disable RTC reference clock detection function
Precondition	-
The called functions	rtc_init_mode_enter/rtc_init_mode_exit
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* disableRTC reference clock detection function*/
ErrStatus error_status = rtc_refclock_detection_disable();
```

rtc_wakeup_enable

The description of `rtc_wakeup_enable` shown as below:

Table 3-661. Function `rtc_wakeup_enable`

Function name	rtc_wakeup_enable
Function prototype	void rtc_wakeup_enable(void);
Function descriptions	enable RTC auto wakeup function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable RTC auto wakeup function */
rtc_wakeup_enable();
```

rtc_wakeup_disable

The description of `rtc_wakeup_disable` shown as below:

Table 3-662. Function `rtc_wakeup_disable`

Function name	rtc_wakeup_disable
Function prototype	ErrStatus rtc_wakeup_disable(void);

Function descriptions	disable RTC auto wakeup function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* disable RTC auto wakeup function */

rtc_wakeup_disable();
```

rtc_wakeup_clock_set

The description of `rtc_wakeup_clock_set` shown as below:

Table 3-663. Function `rtc_wakeup_clock_set`

Function name	rtc_wakeup_clock_set
Function prototype	ErrStatus rtc_wakeup_clock_set(uint8_t wakeup_clock);
Function descriptions	set RTC auto wakeup timer clock
Precondition	-
The called functions	-
Input parameter{in}	
wakeup_clock	Auto-wakeup timert clock selection
WAKEUP_RTCCK_DIV 16	RTC auto wakeup timer clock is RTC clock divided by 16
WAKEUP_RTCCK_DIV 8	RTC auto wakeup timer clock is RTC clock divided by 8
WAKEUP_RTCCK_DIV 4	RTC auto wakeup timer clock is RTC clock divided by 4
WAKEUP_RTCCK_DIV	RTC auto wakeup timer clock is RTC clock divided by 2

2	
WAKEUP_CKSPRE	RTC auto wakeup timer clock is ckspre
WAKEUP_CKSPRE_2 EXP16	RTC auto wakeup timer clock is ckspre and wakeup timer add 2exp16
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* set RTC auto wakeup timer clock */
ErrStatus error_status = rtc_wakeup_clock_set (WAKEUP_RTCCK_DIV16);
```

rtc_wakeup_timer_set

The description of `rtc_wakeup_timer_set` shown as below:

Table 3-664. Function `rtc_wakeup_timer_set`

Function name	rtc_wakeup_timer_set
Function prototype	ErrStatus rtc_wakeup_timer_set(uint16_t wakeup_timer);
Function descriptions	set wakeup timer value
Precondition	-
The called functions	-
Input parameter{in}	
wakeup_timer	Auto_wakeup timer reloads value(0x0000-0xFFFF)
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* set wakeup timer value */
ErrStatus error_status = rtc_wakeup_timer_set (0xFFEE);
```

rtc_wakeup_timer_get

The description of `rtc_wakeup_timer_get` shown as below:

Table 3-665. Function `rtc_wakeup_timer_get`

Function name	rtc_wakeup_timer_get
Function prototype	uint16_t rtc_wakeup_timer_get(void);
Function descriptions	get wakeup timer value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	0-0xFFFF

Example:

```
/* get wakeup timer value */
rtc_wakeup_timer_get();
```

rtc_smooth_calibration_config

The description of `rtc_smooth_calibration_config` shown as below:

Table 3-666. Function `rtc_smooth_calibration_config`

Function name	rtc_smooth_calibration_config
Function prototype	ErrStatus rtc_smooth_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);
Function descriptions	configure RTC smooth calibration
Precondition	-
The called functions	-
Input parameter{in}	
window	select calibration window

<i>RTC_CALIBRATION_WINDOW_32S</i>	2exp20 RTCCLK cycles, 32s if RTCCLK = 32768 Hz
<i>RTC_CALIBRATION_WINDOW_16S</i>	2exp19 RTCCLK cycles, 16s if RTCCLK = 32768 Hz
<i>RTC_CALIBRATION_WINDOW_8S</i>	2exp18 RTCCLK cycles, 8s if RTCCLK = 32768 Hz
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* configure RTC smooth calibration */

ErrStatus error_status;

error_status = rtc_smooth_calibration_config(RTC_CALIBRATION_WINDOW_32S,
                                             RTC_CALIBRATION_PLUS_SET);
```

rtc_coarse_calibration_enable

The description of `rtc_coarse_calibration_enable` shown as below:

Table 3-667. Function `rtc_coarse_calibration_enable`

Function name	rtc_coarse_calibration_enable
Function prototype	ErrStatus rtc_coarse_calibration_enable(void);
Function descriptions	enable RTC coarse calibration
Precondition	-
The called functions	rtc_init_mode_enter/rtc_init_mode_exit
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```

/* enable RTC coarse calibration */

rtc_coarse_calibration_enable();
```

rtc_coarse_calibration_disable

The description of `rtc_coarse_calibration_disable` shown as below:

Table 3-668. Function `rtc_coarse_calibration_disable`

Function name	rtc_coarse_calibration_disable
Function prototype	ErrStatus rtc_coarse_calibration_disable(void);
Function descriptions	disable RTC coarse calibration
Precondition	-
The called functions	rtc_init_mode_enter/rtc_init_mode_exit
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```

/* rtc_coarse_calibration_disable */

ErrStatus error_status = rtc_coarse_calibration_disable();
```

rtc_coarse_calibration_config

The description of `rtc_coarse_calibration_config` shown as below:

Table 3-669. Function `rtc_coarse_calibration_config`

Function name	rtc_coarse_calibration_config
Function prototype	ErrStatus rtc_coarse_calibration_config(uint8_t direction, uint8_t step);
Function descriptions	config coarse calibration direction and step
Precondition	-
The called functions	rtc_init_mode_enter/rtc_init_mode_exit

Input parameter{in}	
direction	coarse calibration direction
<i>CALIB_INCREASE</i>	Increase calendar update frequency
<i>CALIB_DECREASE</i>	Decrease calendar update frequency
Input parameter{in}	
step	Coarse calibration direction
<i>0x00-0x1F</i>	COSD=0: 0x00: +0PPM 0x01: +4PPM(approximate value) 0x02: +8PPM (approximate value) 0x1F: +126PPM (approximate value) COSD=1: 0x00: -0PPM 0x01: -2PPM(approximate value) 0x02: -4PPM (approximate value) 0x1F: -63PPM (approximate value)
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* config coarse calibration direction and step */  
  
ErrStatus error_status = rtc_coarse_calibration_config (INCREASE, 0x01);
```

3.23. SDIO

The secure digital input/output interface (SDIO) defines the SD/SD I/O /MMC CE-ATA card host interface, which provides command/data transfer between the APB2 system bus and SD memory cards, SD I/O cards, Multimedia Card (MMC), and CE-ATA devices. The SDIO registers are listed in chapter [3.23.1](#), the SDIO firmware functions are introduced in chapter [3.23.2](#).

3.23.1. Descriptions of Peripheral registers

SDIO registers are listed in the table shown as below:

Table 3-670. SDIO Registers

Registers	Descriptions
SDIO_PWRCTL	Power control register
SDIO_CLKCTL	Clock control register
SDIO_CMDAGMT	Command argument register
SDIO_CMDCTL	Command control register
SDIO_RSPCMDIDX	Command index response register
SDIO_RESPx x=0..3	Response register
SDIO_DATATO	Data timeout register
SDIO_DATALEN	Data length register
SDIO_DATACTL	Data control register
SDIO_DATACNT	Data counter register
SDIO_STAT	Status register
SDIO_INTC	Interrupt clear register
SDIO_INTEN	Interrupt enable register
SDIO_FIFOCNT	FIFO counter register
SDIO_FIFO	FIFO data register

3.23.2. Descriptions of Peripheral functions

SDIO firmware functions are listed in the table shown as below:

Table 3-671. SDIO firmware function

Function name	Function description
sdio_deinit	deinitialize the SDIO
sdio_clock_config	configure the SDIO clock
sdio_hardware_clock_enable	enable hardware clock control
sdio_hardware_clock_disable	disable hardware clock control

Function name	Function description
sdio_bus_mode_set	set different SDIO card bus mode
sdio_power_state_set	set the SDIO power state
sdio_power_state_get	get the SDIO power state
sdio_clock_enable	enable SDIO_CLK clock output
sdio_clock_disable	disable SDIO_CLK clock output
sdio_command_response_config	configure the command and response
sdio_wait_type_set	set the command state machine wait type
sdio_csm_enable	enable the CSM(command state machine)
sdio_csm_disable	disable the CSM(command state machine)
sdio_command_index_get	get the last response command index
sdio_response_get	get the response for the last received command
sdio_data_config	configure the data timeout, data length and data block size
sdio_data_transfer_config	configure the data transfer mode and direction
sdio_dsm_enable	enable the DSM(data state machine) for data transfer
sdio_dsm_disable	disable the DSM(data state machine)
sdio_data_write	write data(one word) to the transmit FIFO
sdio_data_read	read data(one word) from the receive FIFO
sdio_data_counter_get	get the number of remaining data bytes to be transferred to card
sdio_fifo_counter_get	get the number of words remaining to be written or read from FIFO
sdio_dma_enable	enable the DMA request for SDIO
sdio_dma_disable	disable the DMA request for SDIO
sdio_flag_get	get the flags state of SDIO
sdio_flag_clear	clear the pending flags of SDIO
sdio_interrupt_enable	enable the SDIO interrupt
sdio_interrupt_disable	disable the SDIO interrupt
sdio_interrupt_flag_get	get the interrupt flags state of SDIO

Function name	Function description
sdio_interrupt_flag_clear	clear the interrupt pending flags of SDIO
sdio_readwait_enable	enable the read wait mode(SD I/O only)
sdio_readwait_disable	disable the read wait mode(SD I/O only)
sdio_stop_readwait_enable	enable the function that stop the read wait process(SD I/O only)
sdio_stop_readwait_disable	disable the function that stop the read wait process(SD I/O only)
sdio_readwait_type_set	set the read wait type(SD I/O only)
sdio_operation_enable	enable the SD I/O mode specific operation(SD I/O only)
sdio_operation_disable	disable the SD I/O mode specific operation(SD I/O only)
sdio_suspend_enable	enable the SD I/O suspend operation(SD I/O only)
sdio_suspend_disable	disable the SD I/O suspend operation(SD I/O only)
sdio_ceata_command_enable	enable the CE-ATA command(CE-ATA only)
sdio_ceata_command_disable	disable the CE-ATA command(CE-ATA only)
sdio_ceata_interrupt_enable	enable the CE-ATA interrupt(CE-ATA only)
sdio_ceata_interrupt_disable	disable the CE-ATA interrupt(CE-ATA only)
sdio_ceata_command_completion_enable	enable the CE-ATA command completion signal(CE-ATA only)
sdio_ceata_command_completion_disable	disable the CE-ATA command completion signal(CE-ATA only)
sdio_clock_config	configure the SDIO clock

sdio_deinit

The description of `sdio_deinit` is shown as below:

Table 3-672. Function `sdio_deinit`

Function name	<code>sdio_deinit</code>
Function prototype	<code>void sdio_deinit(void);</code>
Function descriptions	deinitialize the SDIO
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize the SDIO */
sdio_deinit();
```

sdio_clock_config

The description of **sdio_clock_config** is shown as below:

Table 3-673. Function `sdio_clock_config`

Function name	<code>sdio_clock_config</code>
Function prototype	<code>void sdio_clock_config(uint32_t clock_edge, uint32_t clock_bypass, uint32_t clock_powersave, uint16_t clock_division);</code>
Function descriptions	configure the SDIO clock
Precondition	-
The called functions	-
Input parameter{in}	
clock_edge	SDIO_CLK clock edge
SDIO_SDIOCLKEDGE_RISING	select the rising edge of the SDIOCLK to generate SDIO_CLK
SDIO_SDIOCLKEDGE_FALLING	select the falling edge of the SDIOCLK to generate SDIO_CLK
Input parameter{in}	
clock_bypass	clock bypass
SDIO_CLOCKBYPASS_ENABLE	clock bypass

<i>SDIO_CLOCKBYPASS_DISABLE</i>	no bypass
Input parameter{in}	
clock_powersave	SDIO_CLK clock dynamic switch on/off for power saving
<i>SDIO_CLOCKPWRSAVE_ENABLE</i>	SDIO_CLK closed when bus is idle
<i>SDIO_CLOCKPWRSAVE_DISABLE</i>	SDIO_CLK clock is always on
Input parameter{in}	
clock_division	clock division, less than 512
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the SDIO clock */
sdio_clock_config(SDIO_SDIOCLKEDGE_RISING,      SDIO_CLOCKBYPASS_DISABLE,
SDIO_CLOCKPWRSAVE_DISABLE, SD_CLK_DIV_TRANS);
```

sdio_hardware_clock_enable

The description of **sdio_hardware_clock_enable** is shown as below:

Table 3-674. Function *sdio_hardware_clock_enable*

Function name	sdio_hardware_clock_enable
Function prototype	void sdio_hardware_clock_enable(void);
Function descriptions	enable hardware clock control
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable hardware clock control */

sdio_hardware_clock_enable();
```

sdio_hardware_clock_disable

The description of `sdio_hardware_clock_disable` is shown as below:

Table 3-675. Function `sdio_hardware_clock_disable`

Function name	sdio_hardware_clock_disable
Function prototype	void sdio_hardware_clock_disable(void);
Function descriptions	disable hardware clock control
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable hardware clock control */

sdio_hardware_clock_disable();
```

sdio_bus_mode_set

The description of `sdio_bus_mode_set` is shown as below:

Table 3-676. Function `sdio_bus_mode_set`

Function name	sdio_bus_mode_set
Function prototype	void sdio_bus_mode_set(uint32_t bus_mode);

Function descriptions	set different SDIO card bus mode
Precondition	-
The called functions	-
Input parameter{in}	
bus_mode	SDIO card bus mode
<i>SDIO_BUSMODE_1BIT</i> <i>T</i>	1-bit SDIO card bus mode
<i>SDIO_BUSMODE_4BIT</i> <i>T</i>	4-bit SDIO card bus mode
<i>SDIO_BUSMODE_8BIT</i> <i>T</i>	8-bit SDIO card bus mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set SDIO bus mode */
sdio_bus_mode_set(SDIO_BUSMODE_1BIT);
```

sdio_power_state_set

The description of `sdio_power_state_set` is shown as below:

Table 3-677. Function `sdio_power_state_set`

Function name	<code>sdio_power_state_set</code>
Function prototype	<code>void sdio_power_state_set(uint32_t power_state);</code>
Function descriptions	set the SDIO power state
Precondition	-
The called functions	-
Input parameter{in}	
power_state	SDIO power state
<i>SDIO_POWER_ON</i>	SDIO power on

SDIO_POWER_OFF	SDIO power off
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set SDIO power state */

sdio_power_state_set(SDIO_POWER_ON);
```

sdio_power_state_get

The description of `sdio_power_state_get` is shown as below:

Table 3-678. Function `sdio_power_state_get`

Function name	<code>sdio_power_state_get</code>
Function prototype	<code>uint32_t sdio_power_state_get(void);</code>
Function descriptions	get the SDIO power state
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
<code>uint32_t</code>	SDIO_POWER_ON / SDIO_POWER_OFF

Example:

```
/* get the SDIO power state */

uint32_t sdio_power_value;

sdio_power_value = sdio_power_state_get();
```

sdio_clock_enable

The description of `sdio_clock_enable` is shown as below:

Table 3-679. Function sdio_clock_enable

Function name	sdio_clock_enable
Function prototype	void sdio_clock_enable(void);
Function descriptions	enable SDIO_CLK clock output
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SDIO_CLK clock output */

sdio_clock_enable();
```

sdio_clock_disable

The description of sdio_clock_disable is shown as below:

Table 3-680. Function sdio_clock_disable

Function name	sdio_clock_disable
Function prototype	void sdio_clock_disable(void);
Function descriptions	disable SDIO_CLK clock output
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable SDIO_CLK clock output */
sdio_clock_disable();
```

sdio_command_response_config

The description of `sdio_command_response_config` is shown as below:

Table 3-681. Function `sdio_command_response_config`

Function name	sdio_command_response_config
Function prototype	void sdio_command_response_config(uint32_t cmd_index, uint32_t cmd_argument, uint32_t response_type);
Function descriptions	configure the command and response
Precondition	-
The called functions	-
Input parameter{in}	
cmd_index	command index, refer to the related specifications
Input parameter{in}	
cmd_argument	command argument, refer to the related specifications
Input parameter{in}	
response_type	response type
<i>SDIO_RESPONSETYP_E_NO</i>	no response
<i>SDIO_RESPONSETYP_E_SHORT</i>	short response
<i>SDIO_RESPONSETYP_E_LONG</i>	long response
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CMD2(SD_CMD_ALL_SEND_CID) command response config*/
sdio_command_response_config(SD_CMD_ALL_SEND_CID, (uint32_t)0x0,
SDIO_RESPONSETYPE_LONG);
```

sdio_wait_type_set

The description of `sdio_wait_type_set` is shown as below:

Table 3-682. Function `sdio_wait_type_set`

Function name	sdio_wait_type_set
Function prototype	void sdio_wait_type_set(uint32_t wait_type);
Function descriptions	set the command state machine wait type
Precondition	-
The called functions	-
Input parameter{in}	
wait_type	wait type
SDIO_WAITTYPE_NO	not wait interrupt
SDIO_WAITTYPE_INT_ERRUPT	wait interrupt
SDIO_WAITTYPE_DATAEND	wait the end of data transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the command state machine wait type */
sdio_wait_type_set(SDIO_WAITTYPE_NO);
```

sdio_csm_enable

The description of `sdio_csm_enable` is shown as below:

Table 3-683. Function sdio_csm_enable

Function name	sdio_csm_enable
Function prototype	void sdio_csm_enable(void);
Function descriptions	enable the CSM(command state machine)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CSM(command state machine) */

sdio_csm_enable();
```

sdio_csm_disable

The description of sdio_csm_disable is shown as below:

Table 3-684. Function sdio_csm_disable

Function name	sdio_csm_disable
Function prototype	void sdio_csm_disable(void);
Function descriptions	disable the CSM(command state machine)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable the CSM(command state machine) */
sdio_csm_disable();
```

sdio_command_index_get

The description of `sdio_command_index_get` is shown as below:

Table 3-685. Function `sdio_command_index_get`

Function name	sdio_command_index_get
Function prototype	uint8_t sdio_command_index_get(void);
Function descriptions	get the last response command index
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint8_t	last response command index

Example:

```
/* get SDIO command index */
uint8_t sdio_command_value;
sdio_command_value = sdio_command_index_get();
```

sdio_response_get

The description of `sdio_response_get` is shown as below:

Table 3-686. Function `sdio_response_get`

Function name	sdio_response_get
Function prototype	uint32_t sdio_response_get(uint32_t responsex);
Function descriptions	get the response for the last received command

Precondition	-
The called functions	-
Input parameter{in}	
responsex	SDIO response
<i>SDIO_RESPONSE0</i>	card response[31:0]/card response[127:96]
<i>SDIO_RESPONSE1</i>	card response[95:64]
<i>SDIO_RESPONSE2</i>	card response[63:32]
<i>SDIO_RESPONSE3</i>	card response[31:1], plus bit 0
Output parameter{out}	
-	-
Return value	
uint32_t	response for the last received command

Example:

```
/* store the CID0 numbers */

uint32_t sdio_cid[0];

sdio_cid[0] = sdio_response_get(SDIO_RESPONSE0);
```

sdio_data_config

The description of `sdio_data_config` is shown as below:

Table 3-687. Function `sdio_data_config`

Function name	<code>sdio_data_config</code>
Function prototype	<code>void sdio_data_config(uint32_t data_timeout, uint32_t data_length, uint32_t data_blocksize);</code>
Function descriptions	configure the data timeout, data length and data block size
Precondition	-
The called functions	-
Input parameter{in}	
data_timeout	data timeout period in card bus clock periods
Input parameter{in}	

data_length	number of data bytes to be transferred
Input parameter{in}	
data_blocksize	size of data block for block transfer
<i>SDIO_DATABLOCKSIZE_1BYTE</i>	block size = 1 byte
<i>SDIO_DATABLOCKSIZE_2BYTES</i>	block size = 2 bytes
<i>SDIO_DATABLOCKSIZE_4BYTES</i>	block size = 4 bytes
<i>SDIO_DATABLOCKSIZE_8BYTES</i>	block size = 8 bytes
<i>SDIO_DATABLOCKSIZE_16BYTES</i>	block size = 16 bytes
<i>SDIO_DATABLOCKSIZE_32BYTES</i>	block size = 32 bytes
<i>SDIO_DATABLOCKSIZE_64BYTES</i>	block size = 64 bytes
<i>SDIO_DATABLOCKSIZE_128BYTES</i>	block size = 128 bytes
<i>SDIO_DATABLOCKSIZE_256BYTES</i>	block size = 256 bytes
<i>SDIO_DATABLOCKSIZE_512BYTES</i>	block size = 512 bytes
<i>SDIO_DATABLOCKSIZE_1024BYTES</i>	block size = 1024 bytes
<i>SDIO_DATABLOCKSIZE_2048BYTES</i>	block size = 2048 bytes
<i>SDIO_DATABLOCKSIZE_4096BYTES</i>	block size = 4096 bytes
<i>SDIO_DATABLOCKSIZE_8192BYTES</i>	block size = 8192 bytes
<i>SDIO_DATABLOCKSIZE_16384BYTES</i>	block size = 16384 bytes

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SDIO data */
sdio_data_config(0, 0, SDIO_DATABLOCKSIZE_1BYTE);
```

sdio_data_transfer_config

The description of **sdio_data_transfer_config** is shown as below:

Table 3-688. Function `sdio_data_transfer_config`

Function name	<code>sdio_data_transfer_config</code>
Function prototype	<code>void sdio_data_transfer_config(uint32_t transfer_mode, uint32_t transfer_direction);</code>
Function descriptions	configure the data transfer mode and direction
Precondition	-
The called functions	-
Input parameter{in}	
transfer_mode	mode of data transfer
<code>SDIO_TRANSMODE_B LOCK</code>	block transfer
<code>SDIO_TRANSMODE_S TREAM</code>	stream transfer or SDIO multibyte transfer
Input parameter{in}	
transfer_direction	data transfer direction, read or write
<code>SDIO_TRANSDIRECTI ON_TOCARD</code>	write data to card
<code>SDIO_TRANSDIRECTI ON_TOSDIO</code>	read data from card
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure SDIO data transmission */
sdio_data_transfer_config(SDIO_TRANSDIRECTION_TOSDIO,
SDIO_TRANSMODE_BLOCK);
```

sdio_dsm_enable

The description of `sdio_dsm_enable` is shown as below:

Table 3-689. Function `sdio_dsm_enable`

Function name	sdio_dsm_enable
Function prototype	void sdio_dsm_enable(void);
Function descriptions	enable the DSM(data state machine) for data transfer
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the DSM(data state machine) */
sdio_dsm_enable();
```

sdio_dsm_disable

The description of `sdio_dsm_disable` is shown as below:

Table 3-690. Function `sdio_dsm_disable`

Function name	sdio_dsm_disable
Function prototype	void sdio_dsm_disable(void);

Function descriptions	disable the DSM(data state machine)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the DSM(data state machine) */

sdio_dsm_disable();
```

sdio_data_write

The description of sdio_data_write is shown as below:

Table 3-691. Function sdio_data_write

Function name	sdio_data_write
Function prototype	void sdio_data_write(uint32_t data);
Function descriptions	write data(one word) to the transmit FIFO
Precondition	-
The called functions	-
Input parameter{in}	
data	32-bit data write to card
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write data(one word) to the transmit FIFO */
```

```
sdio_data_write(0x0000 0001);
```

sdio_data_read

The description of `sdio_data_read` is shown as below:

Table 3-692. Function `sdio_data_read`

Function name	<code>sdio_data_read</code>
Function prototype	<code>uint32_t sdio_data_read(void);</code>
Function descriptions	read data(one word) from the receive FIFO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
<code>uint32_t</code>	received data

Example:

```
/* read data(one word) from the receive FIFO */
sdio_data_read();
```

sdio_data_counter_get

The description of `sdio_data_counter_get` is shown as below:

Table 3-693. Function `sdio_data_counter_get`

Function name	<code>sdio_data_counter_get</code>
Function prototype	<code>uint32_t sdio_data_counter_get(void);</code>
Function descriptions	get the number of remaining data bytes to be transferred to card
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
uint32_t	number of remaining data bytes to be transferred

Example:

```

/* get the number of remaining data bytes to be transferred to card */

uint32_t sdio_data_value;

sdio_data_value = sdio_data_counter_get();

```

sdio_fifo_counter_get

The description of `sdio_fifo_counter_get` is shown as below:

Table 3-694. Function `sdio_data_counter_get`

Function name	sdio_fifo_counter_get
Function prototype	uint32_t sdio_fifo_counter_get(void);
Function descriptions	get the number of words remaining to be written or read from FIFO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	remaining number of words

Example:

```

/* get the number of words remaining to be written or read from FIFO */

uint32_t sdio_fifo_value;

sdio_fifo_value = sdio_fifo_counter_get();

```

sdio_dma_enable

The description of `sdio_dma_enable` is shown as below:

Table 3-695. Function sdio_dma_enable

Function name	sdio_dma_enable
Function prototype	void sdio_dma_enable(void);
Function descriptions	enable the DMA request for SDIO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the SDIO DMA */
sdio_dma_enable();
```

sdio_dma_disable

The description of sdio_dma_disable is shown as below:

Table 3-696. Function sdio_dma_disable

Function name	sdio_dma_disable
Function prototype	void sdio_dma_disable(void);
Function descriptions	disable the DMA request for SDIO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable the SDIO DMA */
sdio_dma_disable();
```

sdio_flag_get

The description of `sdio_flag_get` is shown as below:

Table 3-697. Function `sdio_flag_get`

Function name	sdio_flag_get
Function prototype	FlagStatus sdio_flag_get(uint32_t flag);
Function descriptions	get the flags state of SDIO
Precondition	-
The called functions	-
Input parameter{in}	
flag	flags state of SDIO
<i>SDIO_FLAG_CCRCER</i> <i>R</i>	command response received (CRC check failed) flag
<i>SDIO_FLAG_DTCRCE</i> <i>RR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_FLAG_CMDTMO</i> <i>UT</i>	command response timeout flag
<i>SDIO_FLAG_DTTMOU</i> <i>T</i>	data timeout flag
<i>SDIO_FLAG_TXURE</i>	transmit FIFO underrun error occurs flag
<i>SDIO_FLAG_RXORE</i>	received FIFO overrun error occurs flag
<i>SDIO_FLAG_CMDREC</i> <i>V</i>	command response received (CRC check passed) flag
<i>SDIO_FLAG_CMDSEN</i> <i>D</i>	command sent (no response required) flag
<i>SDIO_FLAG_DTEND</i>	data end (data counter, SDIO_DATACNT, is zero) flag
<i>SDIO_FLAG_STBITE</i>	start bit error in the bus flag

<i>SDIO_FLAG_DTBKLE_ND</i>	data block sent/received (CRC check passed) flag
<i>SDIO_FLAG_CMDRUN</i>	command transmission in progress flag
<i>SDIO_FLAG_TXRUN</i>	data transmission in progress flag
<i>SDIO_FLAG_RXRUN</i>	data reception in progress flag
<i>SDIO_FLAG_TFH</i>	transmit FIFO is half empty flag: at least 8 words can be written into the FIFO
<i>SDIO_FLAG_RFH</i>	receive FIFO is half full flag: at least 8 words can be read in the FIFO
<i>SDIO_FLAG_TFF</i>	transmit FIFO is full flag
<i>SDIO_FLAG_RFF</i>	receive FIFO is full flag
<i>SDIO_FLAG_TFE</i>	transmit FIFO is empty flag
<i>SDIO_FLAG_RFE</i>	receive FIFO is empty flag
<i>SDIO_FLAG_TXDTVAL</i>	data is valid in transmit FIFO flag
<i>SDIO_FLAG_RXDTVAL</i>	data is valid in receive FIFO flag
<i>SDIO_FLAG_SDIOINT</i>	SD I/O interrupt received flag
<i>SDIO_FLAG_ATAEND</i>	CE-ATA command completion signal received (only for CMD61) flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the flags state of SDIO */

FlagStatus flag_value;

flag_value = sdio_flag_get(SDIO_FLAG_RFH);
```

sdio_flag_clear

The description of **sdio_flag_clear** is shown as below:

Table 3-698. Function *sdio_flag_clear*

Function name	<i>sdio_flag_clear</i>

Function prototype	void sdio_flag_clear(uint32_t flag);
Function descriptions	clear the pending flags of SDIO
Precondition	-
The called functions	-
Input parameter{in}	
flag	flags state of SDIO
<i>SDIO_FLAG_CCRKER</i> <i>R</i>	command response received (CRC check failed) flag
<i>SDIO_FLAG_DTCRCE</i> <i>RR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_FLAG_CMDTMO</i> <i>UT</i>	command response timeout flag
<i>SDIO_FLAG_DTTMOU</i> <i>T</i>	data timeout flag
<i>SDIO_FLAG_TXURE</i>	transmit FIFO underrun error occurs flag
<i>SDIO_FLAG_RXORE</i>	received FIFO overrun error occurs flag
<i>SDIO_FLAG_CMDREC</i> <i>V</i>	command response received (CRC check passed) flag
<i>SDIO_FLAG_CMDSEN</i> <i>D</i>	command sent (no response required) flag
<i>SDIO_FLAG_DTEND</i>	data end (data counter, SDIO_DATACNT, is zero) flag
<i>SDIO_FLAG_STBITE</i>	start bit error in the bus flag
<i>SDIO_FLAG_DTBLKE</i> <i>ND</i>	data block sent/received (CRC check passed) flag
<i>SDIO_FLAG_SDIOINT</i>	SD I/O interrupt received flag
<i>SDIO_FLAG_ATAEND</i>	CE-ATA command completion signal received (only for CMD61) flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the pending flags of SDIO */
sdio_flag_clear(SDIO_FLAG_DTCRCERR);
```

sdio_interrupt_enable

The description of `sdio_interrupt_enable` is shown as below:

Table 3-699. Function `sdio_interrupt_enable`

Function name	sdio_interrupt_enable
Function prototype	void sdio_interrupt_enable(uint32_t int_flag);
Function descriptions	enable the SDIO interrupt
Precondition	-
The called functions	-
Input parameter{in}	
<i>int_flag</i>	interrupt flags state of SDIO
<i>SDIO_INT_CCRCERR</i>	SDIO CCRCERR interrupt
<i>SDIO_INT_DTCRCER</i> <i>R</i>	SDIO DTCRCERR interrupt
<i>SDIO_INT_CMDTMOU</i> <i>T</i>	SDIO CMDTMOU interrupt
<i>SDIO_INT_DTTMOUT</i>	SDIO DTTMOUT interrupt
<i>SDIO_INT_TXURE</i>	SDIO TXURE interrupt
<i>SDIO_INT_RXORE</i>	SDIO_RXORE
<i>SDIO_INT_CMDRECV</i>	SDIO CMDRECV interrupt
<i>SDIO_INT_CMDSEND</i>	SDIO CMDSEND interrupt
<i>SDIO_INT_DTEND</i>	SDIO DTEND interrupt
<i>SDIO_INT_STBITE</i>	SDIO STBITE interrupt
<i>SDIO_INT_DTBLKEND</i>	SDIO DTBLKEND interrupt
<i>SDIO_INT_CMDRUN</i>	SDIO CMDRUN interrupt
<i>SDIO_INT_TXRUN</i>	SDIO TXRUN interrupt
<i>SDIO_INT_RXRUN</i>	SDIO RXRUN interrupt
<i>SDIO_INT_TFH</i>	SDIO TFH interrupt

<i>SDIO_INT_RFH</i>	SDIO RFH interrupt
<i>SDIO_INT_TFF</i>	SDIO TFF interrupt
<i>SDIO_INT_RFF</i>	SDIO RFF interrupt
<i>SDIO_INT_TFE</i>	SDIO TFE interrupt
<i>SDIO_INT_RFE</i>	SDIO RFE interrupt
<i>SDIO_INT_TXDTVAL</i>	SDIO TXDTVAL interrupt
<i>SDIO_INT_RXDTVAL</i>	SDIO RXDTVAL interrupt
<i>SDIO_INT_SDIOINT</i>	SDIO SDIOINT interrupt
<i>SDIO_INT_ATAEND</i>	SDIO ATAEND interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the SDIO corresponding interrupts */

sdio_interrupt_enable(SDIO_INT_CCRCERR | SDIO_INT_DTTMOUT | SDIO_INT_RXORE
| SDIO_INT_DTEND | SDIO_INT_STBITE);
```

sdio_interrupt_disable

The description of `sdio_interrupt_disable` is shown as below:

Table 3-700. Function `sdio_interrupt_disable`

Function name	<code>sdio_interrupt_disable</code>
Function prototype	<code>void sdio_interrupt_disable(uint32_t int_flag);</code>
Function descriptions	disable the SDIO interrupt
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	interrupt flags state of SDIO
<i>SDIO_INT_CCRCERR</i>	SDIO CCRCERR interrupt

<i>SDIO_INT_DTCRCER</i>	R	SDIO DTCRCERR interrupt
<i>SDIO_INT_CMDTMOUT</i>	T	SDIO CMDTMOUT interrupt
<i>SDIO_INT_DTTMOUT</i>		SDIO DTTMOUT interrupt
<i>SDIO_INT_TXURE</i>		SDIO TXURE interrupt
<i>SDIO_INT_RXORE</i>		SDIO_INT_RXORE
<i>SDIO_INT_CMDRECV</i>		SDIO CMDRECV interrupt
<i>SDIO_INT_CMDSEND</i>		SDIO CMDSEND interrupt
<i>SDIO_INT_DTEND</i>		SDIO DTEND interrupt
<i>SDIO_INT_STBITE</i>		SDIO STBITE interrupt
<i>SDIO_INT_DTBLKEND</i>		SDIO DTBLKEND interrupt
<i>SDIO_INT_CMDRUN</i>		SDIO CMDRUN interrupt
<i>SDIO_INT_TXRUN</i>		SDIO TXRUN interrupt
<i>SDIO_INT_RXRUN</i>		SDIO RXRUN interrupt
<i>SDIO_INT_TFH</i>		SDIO TFH interrupt
<i>SDIO_INT_RFH</i>		SDIO RFH interrupt
<i>SDIO_INT_TFF</i>		SDIO TFF interrupt
<i>SDIO_INT_RFF</i>		SDIO RFF interrupt
<i>SDIO_INT_TFE</i>		SDIO TFE interrupt
<i>SDIO_INT_RFE</i>		SDIO RFE interrupt
<i>SDIO_INT_TXDTVAL</i>		SDIO TXDTVAL interrupt
<i>SDIO_INT_RXDTVAL</i>		SDIO RXDTVAL interrupt
<i>SDIO_INT_SDIOINT</i>		SDIO SDIOINT interrupt
<i>SDIO_INT_ATAEND</i>		SDIO ATAEND interrupt
Output parameter{out}		
-		-
Return value		
-		-

Example:

```

/* disable the SDIO interrupt */

sdio_interrupt_disable(SDIO_INT_DTCRCERR);

```

sdio_interrupt_flag_get

The description of `sdio_interrupt_flag_get` is shown as below:

Table 3-701. Function `sdio_interrupt_flag_get`

Function name	<code>sdio_interrupt_flag_get</code>
Function prototype	<code>FlagStatus sdio_interrupt_flag_get(uint32_t int_flag);</code>
Function descriptions	get the interrupt flags state of SDIO
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	interrupt flags state of SDIO
<code>SDIO_INT_FLAG_CCR CERR</code>	SDIO CCRCERR interrupt flag
<code>SDIO_INT_FLAG_DTC RCERR</code>	SDIO DTCRCERR interrupt flag
<code>SDIO_INT_FLAG_CMD TMOUT</code>	SDIO CMDTMOUT interrupt flag
<code>SDIO_INT_FLAG_DTT MOUT</code>	SDIO DTTMOUT interrupt flag
<code>SDIO_INT_FLAG_TXU RE</code>	SDIO TXURE interrupt flag
<code>SDIO_INT_FLAG_RXO RE</code>	SDIO_INT_RXORE flag
<code>SDIO_INT_FLAG_CMD RECV</code>	SDIO CMDRECV interrupt flag
<code>SDIO_INT_FLAG_CMD SEND</code>	SDIO CMDSEND interrupt flag
<code>SDIO_INT_FLAG_DTE ND</code>	SDIO DTEND interrupt flag

<i>SDIO_INT_FLAG_STBITE</i>	SDIO STBITE interrupt flag
<i>SDIO_INT_FLAG_DTBKEND</i>	SDIO DTBLKEND interrupt flag
<i>SDIO_INT_FLAG_CMDRUN</i>	SDIO CMDRUN interrupt flag
<i>SDIO_INT_FLAG_TXRUN</i>	SDIO TXRUN interrupt flag
<i>SDIO_INT_FLAG_RXRUN</i>	SDIO RXRUN interrupt flag
<i>SDIO_INT_FLAG_TFH</i>	SDIO TFH interrupt flag
<i>SDIO_INT_FLAG_RFH</i>	SDIO RFH interrupt flag
<i>SDIO_INT_FLAG_TFF</i>	SDIO TFF interrupt flag
<i>SDIO_INT_FLAG_RFF</i>	SDIO RFF interrupt flag
<i>SDIO_INT_FLAG_TFE</i>	SDIO TFE interrupt flag
<i>SDIO_INT_FLAG_RFE</i>	SDIO RFE interrupt flag
<i>SDIO_INT_FLAG_TXDTVAL</i>	SDIO TXDTVAL interrupt flag
<i>SDIO_INT_FLAG_RXDTVAL</i>	SDIO RXDTVAL interrupt flag
<i>SDIO_INT_FLAG_SDIOINT</i>	SDIO SDIOINT interrupt flag
<i>SDIO_INT_FLAG_ATAEND</i>	SDIO ATAEND interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the interrupt flags state of SDIO */
```

```
FlagStatus flag_value;
```

```
flag_value = sdio_interrupt_flag_get(SDIO_INT_FLAG_DTEND);
```

sdio_interrupt_flag_clear

The description of `sdio_interrupt_flag_clear` is shown as below:

Table 3-702. Function `sdio_interrupt_flag_clear`

Function name	<code>sdio_interrupt_flag_clear</code>
Function prototype	<code>void sdio_interrupt_flag_clear(uint32_t int_flag);</code>
Function descriptions	clear the interrupt pending flags of SDIO
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	interrupt flags state of SDIO
<i>SDIO_INT_FLAG_CCR</i> <i>CERR</i>	command response received (CRC check failed) flag
<i>SDIO_INT_FLAG_DTC</i> <i>RCERR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_INT_FLAG_CMD</i> <i>TMOUT</i>	command response timeout flag
<i>SDIO_INT_FLAG_DTT</i> <i>MOUT</i>	data timeout flag
<i>SDIO_INT_FLAG_TXU</i> <i>RE</i>	transmit FIFO underrun error occurs flag
<i>SDIO_INT_FLAG_RXO</i> <i>RE</i>	received FIFO overrun error occurs flag
<i>SDIO_INT_FLAG_CMD</i> <i>RECV</i>	command response received (CRC check passed) flag
<i>SDIO_INT_FLAG_CMD</i> <i>SEND</i>	command sent (no response required) flag
<i>SDIO_INT_FLAG_DTE</i> <i>ND</i>	data end (data counter, SDIO_DATACNT, is zero) flag
<i>SDIO_INT_FLAG_STB</i> <i>TE</i>	start bit error in the bus flag
<i>SDIO_INT_FLAG_DTB</i>	data block sent/received (CRC check passed) flag

<i>LKEND</i>	
<i>SDIO_INT_FLAG_SDIOINT</i>	SD I/O interrupt received flag
<i>SDIO_INT_FLAG_ATAEND</i>	CE-ATA command completion signal received (only for CMD61) flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the interrupt pending flags of SDIO */
sdio_interrupt_flag_clear(SDIO_INT_FLAG_DTEND);
```

sdio_readwait_enable

The description of `sdio_readwait_enable` is shown as below:

Table 3-703. Function `sdio_readwait_enable`

Function name	<code>sdio_readwait_enable</code>
Function prototype	<code>void sdio_readwait_enable(void);</code>
Function descriptions	enable the read wait mode(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the read wait mode(SD I/O only) */
sdio_readwait_enable();
```

sdio_readwait_disable

The description of `sdio_readwait_disable` is shown as below:

Table 3-704. Function `sdio_readwait_disable`

Function name	sdio_readwait_disable
Function prototype	<code>void sdio_readwait_disable(void);</code>
Function descriptions	disable the read wait mode(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the read wait mode(SD I/O only) */

sdio_readwait_disable();
```

sdio_stop_readwait_enable

The description of `sdio_stop_readwait_enable` is shown as below:

Table 3-705. Function `sdio_stop_readwait_enable`

Function name	sdio_stop_readwait_enable
Function prototype	<code>void sdio_stop_readwait_enable(void);</code>
Function descriptions	enable the function that stop the read wait process(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable the function that stop the read wait process(SD I/O only) */
sdio_stop_readwait_enable();
```

sdio_stop_readwait_disable

The description of `sdio_stop_readwait_disable` is shown as below:

Table 3-706. Function `sdio_stop_readwait_disable`

Function name	sdio_stop_readwait_disable
Function prototype	void sdio_stop_readwait_disable(void);
Function descriptions	disable the function that stop the read wait process(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the function that stop the read wait process(SD I/O only) */
sdio_stop_readwait_disable();
```

sdio_readwait_type_set

The description of `sdio_readwait_type_set` is shown as below:

Table 3-707. Function `sdio_readwait_type_set`

Function name	sdio_readwait_type_set
Function prototype	void sdio_readwait_type_set(uint32_t readwait_type);

Function descriptions	set the read wait type(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
readwait_type	SD I/O read wait type
<i>SDIO_READWAITTYP_E_CLK</i>	read wait control by stopping SDIO_CLK
<i>SDIO_READWAITTYP_E_DAT2</i>	read wait control using SDIO_DAT[2]
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the read wait type(SD I/O only) */
sdio_readwait_type_set(uint32_t readwait_type);
```

sdio_operation_enable

The description of `sdio_operation_enable` is shown as below:

Table 3-708. Function `sdio_operation_enable`

Function name	sdio_operation_enable
Function prototype	void sdio_operation_enable(void);
Function descriptions	enable the SD I/O mode specific operation(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable the SD I/O mode specific operation(SD I/O only) */

sdio_operation_enable();
```

sdio_operation_disable

The description of `sdio_operation_disable` is shown as below:

Table 3-709. Function `sdio_operation_disable`

Function name	sdio_operation_disable
Function prototype	void sdio_operation_disable(void);
Function descriptions	disable the SD I/O mode specific operation(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the SD I/O mode specific operation(SD I/O only) */

void sdio_operation_disable();
```

sdio_suspend_enable

The description of `sdio_suspend_enable` is shown as below:

Table 3-710. Function `sdio_suspend_enable`

Function name	sdio_suspend_enable
Function prototype	void sdio_suspend_enable(void);
Function descriptions	enable the SD I/O suspend operation(SD I/O only)
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the SD I/O suspend operation(SD I/O only) */
sdio_suspend_enable();
```

sdio_suspend_disable

The description of `sdio_suspend_disable` is shown as below:

Table 3-711. Function `sdio_suspend_disable`

Function name	sdio_suspend_disable
Function prototype	void sdio_suspend_disable(void);
Function descriptions	disable the SD I/O suspend operation(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the SD I/O suspend operation(SD I/O only) */
sdio_suspend_disable();
```

sdio_ceata_command_enable

The description of `sdio_ceata_command_enable` is shown as below:

Table 3-712. Function `sdio_ceata_command_enable`

Function name	<code>sdio_ceata_command_enable</code>
Function prototype	<code>void sdio_ceata_command_enable(void);</code>
Function descriptions	enable the CE-ATA command(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CE-ATA command(CE-ATA only) */
sdio_ceata_command_enable();
```

sdio_ceata_command_disable

The description of `sdio_ceata_command_disable` is shown as below:

Table 3-713. Function `sdio_ceata_command_disable`

Function name	<code>sdio_ceata_command_disable</code>
Function prototype	<code>void sdio_ceata_command_disable(void);</code>
Function descriptions	disable the CE-ATA command(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable the CE-ATA command(CE-ATA only) */

sdio_ceata_command_disable();
```

sdio_ceata_interrupt_enable

The description of `sdio_ceata_interrupt_enable` is shown as below:

Table 3-714. Function `sdio_ceata_interrupt_enable`

Function name	sdio_ceata_interrupt_enable
Function prototype	void sdio_ceata_interrupt_enable(void);
Function descriptions	enable the CE-ATA interrupt(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CE-ATA interrupt(CE-ATA only) */

sdio_ceata_interrupt_enable();
```

sdio_ceata_interrupt_disable

The description of `sdio_ceata_interrupt_disable` is shown as below:

Table 3-715. Function `sdio_ceata_interrupt_disable`

Function name	sdio_ceata_interrupt_disable
Function prototype	void sdio_ceata_interrupt_disable(void);

Function descriptions	disable the CE-ATA interrupt(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CE-ATA interrupt(CE-ATA only) */
sdio_ceata_interrupt_disable();
```

sdio_ceata_command_completion_enable

The description of `sdio_ceata_command_completion_enable` is shown as below:

Table 3-716. Function `sdio_ceata_command_completion_enable`

Function name	sdio_ceata_command_completion_enable
Function prototype	void sdio_ceata_command_completion_enable(void);
Function descriptions	enable the CE-ATA command completion signal(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CE-ATA command completion signal(CE-ATA only) */
```

```
sdio_ceata_command_completion_enable();
```

sdio_ceata_command_completion_disable

The description of sdio_ceata_command_completion_disable is shown as below:

Table 3-717. Function sdio_ceata_command_completion_disable

Function name	sdio_ceata_command_completion_disable
Function prototype	void sdio_ceata_command_completion_disable(void);
Function descriptions	disable the CE-ATA command completion signal(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CE-ATA command completion signal(CE-ATA only) */
sdio_ceata_command_completion_disable();
```

3.24. SPI

The SPI/I2S module can communicate with external devices using the SPI protocol or the I2S audio protocol. The SPI/I2S registers are listed in chapter [3.24.1](#), the SPI/I2S firmware functions are introduced in chapter [3.24.2](#).

3.24.1. Descriptions of Peripheral registers

SPI/I2S registers are listed in the table shown as below:

Table 3-718. SPI/I2S Registers

Registers	Descriptions
SPI_CTL0	SPI control register 0
SPI_CTL1	SPI control register 1
SPI_STAT	SPI status register

Registers	Descriptions
SPI_DATA	SPI data register
SPI_CRCPOLY	SPI CRC polynomial register
SPI_RCRC	SPI receive CRC register
SPI_TCRC	SPI transmit CRC register
SPI_I2SCTL	SPI/I2S control register
SPI_I2SPSC	SPI/I2S clock prescaler register
SPI_QCTL	Quad-SPI mode control register
I2S_ADD_CTL0	I2S_ADD control register 0
I2S_ADD_CTL1	I2S_ADD control register 1
I2S_ADD_STAT	I2S_ADD status register
I2S_ADD_DATA	I2S_ADD data register
I2S_ADD_CRCPOLY	I2S_ADD CRC polynomial register
I2S_ADD_RCRC	I2S_ADD receive CRC register
I2S_ADD_TCRC	I2S_ADD transmit CRC register
I2S_ADD_I2SCTL	I2S_ADD I2S control register
I2S_ADD_I2SPSC	I2S_ADD I2S clock prescaler register

3.24.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

Table 3-719. SPI/I2S firmware function

Function name	Function description
spi_i2s_deinit	reset SPI and I2S peripheral
spi_struct_para_init	initialize the parameters of SPI struct with the default values
spi_init	initialize SPI peripheral parameter
spi_enable	enable SPI
spi_disable	disable SPI
i2s_init	initialize I2S peripheral parameter
i2s_psc_config	configure I2S peripheral prescaler
i2s_enable	enable I2S
i2s_disable	disable I2S
spi_nss_output_enable	enable SPI NSS output function
spi_nss_output_disable	disable SPI NSS output function
spi_nss_internal_high	SPI NSS pin high level in software mode
spi_nss_internal_low	SPI NSS pin low level in software mode
spi_dma_enable	enable SPI DMA function
spi_dma_disable	disable SPI DMA function
spi_i2s_data_frame_format_config	configure SPI/I2S data frame format
spi_i2s_data_transmit	SPI transmit data
spi_i2s_data_receive	SPI receive data
spi_bidirectional_transfer_config	configure SPI bidirectional transfer direction

Function name	Function description
i2s_full_duplex_mode_config	configure i2s full duplex mode
spi_i2s_format_error_clear	clear TI mode format error flag status
spi_crc_polynomial_set	set SPI CRC polynomial
spi_crc_polynomial_get	get SPI CRC polynomial
spi_crc_on	turn on SPI CRC function
spi_crc_off	turn off SPI CRC function
spi_crc_next	SPI next data is CRC value
spi_crc_get	get SPI CRC send value or receive value
spi_crc_error_clear	clear SPI CRC error flag status
spi_ti_mode_enable	enable SPI TI mode
spi_ti_mode_disable	disable SPI TI mode
spi_quad_enable	enable quad wire SPI
spi_quad_disable	disable quad wire SPI
spi_quad_write_enable	enable quad wire SPI write
spi_quad_read_enable	enable quad wire SPI read
spi_quad_io23_output_enable	enable quad wire SPI_IO2 and SPI_IO3 pin output
spi_quad_io23_output_disable	disable quad wire SPI_IO2 and SPI_IO3 pin output
spi_i2s_flag_get	get SPI and I2S flag status
spi_i2s_interrupt_enable	enable SPI and I2S interrupt
spi_i2s_interrupt_disable	disable SPI and I2S interrupt
spi_i2s_interrupt_flag_get	get SPI and I2S interrupt status

Structure spi_parameter_struct

Table 3-720. spi_parameter_struct

Member name	Function description
device_mode	SPI master or slave (SPI_MASTER, SPI_SLAVE)
trans_mode	SPI transtype (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	SPI frame size (SPI_FRAMESIZE_16BIT, SPI_FRAMESIZE_8BIT)
nss	SPI NSS control by hardware or software (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	SPI big endian or little endian (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	SPI clock phase and polarity (SPI_CK_PL_LOW_PH_1EDGE, SPI_CK_PL_HIGH_PH_1EDGE, SPI_CK_PL_LOW_PH_2EDGE, SPI_CK_PL_HIGH_PH_2EDGE)
prescale	SPI prescale factor (SPI_PSC_n (n=2,4,8,16,32,64,128,256))

spi_i2s_deinit

The description of spi_i2s_deinit is shown as below:

Table 3-721. Function spi_i2s_deinit

Function name	spi_i2s_deinit
Function prototype	void spi_i2s_deinit(uint32_t spi_periph);
Function descriptions	reset SPI and I2S peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
spi_periph	SPI/I2S peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset SPI0 */
spi_i2s_deinit(SPI0);
```

spi_struct_para_init

The description of spi_struct_para_init is shown as below:

Table 3-722. Function spi_struct_para_init

Function name	spi_struct_para_init
Function prototype	void spi_struct_para_init(spi_parameter_struct* spi_struct);
Function descriptions	initialize the parameters of SPI struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
*spi_struct	a spi_parameter_struct address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the parameters of SPI */
spi_parameter_struct spi_init_struct;
spi_struct_para_init(&spi_init_struct);
```

spi_init

The description of spi_init is shown as below:

Table 3-723. Function spi_init

Function name	spi_init
Function prototype	void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
Function descriptions	initialize SPI peripheral parameter
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Input parameter{in}	
spi_struct	SPI parameter initialization stuct, the structure members can refer to members of the structure Table 3-720. spi_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize SPI0 */

spi_parameter_struct spi_init_struct;

spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode    = SPI_MASTER;
spi_init_struct.frame_size     = SPI_FRAMESIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss            = SPI_NSS_SOFT;
spi_init_struct.prescale       = SPI_PSC_8;
spi_init_struct.endian         = SPI_ENDIAN_MSB;

spi_init(SPI0, &spi_init_struct);
```

spi_enable

The description of spi_enable is shown as below:

Table 3-724. Function spi_enable

Function name	spi_enable
Function prototype	void spi_enable(uint32_t spi_periph);

Function descriptions	enable SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 */

spi_enable(SPI0);
```

spi_disable

The description of spi_disable is shown as below:

Table 3-725. Function spi_disable

Function name	spi_disable
Function prototype	void spi_disable(uint32_t spi_periph);
Function descriptions	disable SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 */

spi_disable(SPI0);
```

i2s_init

The description of i2s_init is shown as below:

Table 3-726. Function i2s_init

Function name	i2s_init
----------------------	----------

Function prototype	void i2s_init(uint32_t spi_periph,uint32_t i2s_mode, uint32_t i2s_standard, uint32_t i2s_ckpl);
Function descriptions	initialize I2S peripheral parameter
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	I2S peripheral
<i>SPIx</i>	x=1,2
Input parameter{in}	
i2s_mode	I2S operation mode
<i>I2S_MODE_SLAVE TX</i>	I2S slave transmit mode
<i>I2S_MODE_SLAVE RX</i>	I2S slave receive mode
<i>I2S_MODE_MASTER TX</i>	I2S master transmit mode
<i>I2S_MODE_MASTER RX</i>	I2S master receive mode
Input parameter{in}	
i2s_standard	I2S standard
<i>I2S_STD_PHILLIPS</i>	I2S phillips standard
<i>I2S_STD_MSB</i>	I2S MSB standard
<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCMSHORT</i>	I2S PCM short standard
<i>I2S_STD_PCMLONG</i>	I2S PCM long standard
Input parameter{in}	
i2s_ckpl	I2S idle state clock polarity
<i>I2S_CKPL_LOW</i>	I2S clock polarity low level
<i>I2S_CKPL_HIGH</i>	I2S clock polarity high level
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize I2S1 */

i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

i2s_psc_config

The description of i2s_psc_config is shown as below:

Table 3-727. Function i2s_psc_config

Function name	i2s_psc_config
Function prototype	void i2s_psc_config(uint32_t spi_periph, uint32_t i2s_audiosample, uint32_t

	i2s_frameformat, uint32_t i2s_mckout);
Function descriptions	configure I2S prescaler
Precondition	-
The called functions	rcu_i2s_clock_config/ rcu_osc_on/ rcu_osc_stab_wait
Input parameter{in}	
spi_periph	I2S peripheral
SPIx	x=1,2
Input parameter{in}	
i2s_audiosample	I2S audio sample rate
I2S_AUDIOSAMPLE_8K	audio sample rate is 8KHz
I2S_AUDIOSAMPLE_11K	audio sample rate is 11KHz
I2S_AUDIOSAMPLE_16K	audio sample rate is 16KHz
I2S_AUDIOSAMPLE_22K	audio sample rate is 22KHz
I2S_AUDIOSAMPLE_32K	audio sample rate is 32KHz
I2S_AUDIOSAMPLE_44K	audio sample rate is 44KHz
I2S_AUDIOSAMPLE_48K	audio sample rate is 48KHz
I2S_AUDIOSAMPLE_96K	audio sample rate is 96KHz
I2S_AUDIOSAMPLE_192K	audio sample rate is 192KHz
Input parameter{in}	
i2s_frameformat	I2S data length and channel length
I2S_FRAMEFORMAT_DT16B_CH16B	I2S data length is 16 bit and channel length is 16 bit
I2S_FRAMEFORMAT_DT16B_CH32B	I2S data length is 16 bit and channel length is 32 bit
I2S_FRAMEFORMAT_DT24B_CH32B	I2S data length is 24 bit and channel length is 32 bit
I2S_FRAMEFORMAT_DT32B_CH32B	I2S data length is 32 bit and channel length is 32 bit
Input parameter{in}	
i2s_mckout	I2S master clock output
I2S_MCKOUT_ENABLE	I2S master clock output enable
I2S_MCKOUT_DISABLE	I2S master clock output disable

<i>E</i>	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure I2S1 prescaler */

i2s_psc_config(SPI1, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,
I2S_MCKOUT_DISABLE);
```

i2s_enable

The description of i2s_enable is shown as below:

Table 3-728. Function i2s_enable

Function name	i2s_enable
Function prototype	void i2s_enable(uint32_t spi_periph);
Function descriptions	enable I2S
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	I2S peripheral
SPIx	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2S1*/

i2s_enable(SPI1);
```

i2s_disable

The description of i2s_disable is shown as below:

Table 3-729. Function i2s_disable

Function name	i2s_disable
Function prototype	void i2s_disable(uint32_t spi_periph);
Function descriptions	disable I2S
Precondition	-
The called functions	-

Input parameter{in}	
spi_periph	I2S peripheral
SPIx	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2S1*/
i2s_disable(SPI1);
```

spi_nss_output_enable

The description of spi_nss_output_enable is shown as below:

Table 3-730. Function spi_nss_output_enable

Function name	spi_nss_output_enable
Function prototype	void spi_nss_output_enable(uint32_t spi_periph);
Function descriptions	enable SPI NSS output function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 NSS output */
spi_nss_output_enable(SPI0);
```

spi_nss_output_disable

The description of spi_nss_output_disable is shown as below:

Table 3-731. Function spi_nss_output_disable

Function name	spi_nss_output_disable
Function prototype	void spi_nss_output_disable(uint32_t spi_periph);
Function descriptions	disable SPI NSS output function
Precondition	-

The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 NSS output */

spi_nss_output_disable(SPI0);
```

spi_nss_internal_high

The description of **spi_nss_internal_high** is shown as below:

Table 3-732. Function spi_nss_internal_high

Function name	spi_nss_internal_high
Function prototype	void spi_nss_internal_high(uint32_t spi_periph);
Function descriptions	SPI NSS pin high level in software mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 NSS pin is pulled high level in software mode */

spi_nss_internal_high(SPI0);
```

spi_nss_internal_low

The description of **spi_nss_internal_low** is shown as below:

Table 3-733. Function spi_nss_internal_low

Function name	spi_nss_internal_low
Function prototype	void spi_nss_internal_low(uint32_t spi_periph);
Function descriptions	SPI NSS pin low level in software mode

Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 NSS pin is pulled low level in software mode */
spi_nss_internal_low(SPI0);
```

spi_dma_enable

The description of spi_dma_enable is shown as below:

Table 3-734. Function spi_dma_enable

Function name	spi_dma_enable
Function prototype	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
Function descriptions	enable SPI DMA function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Input parameter{in}	
dma	SPI DMA mode
SPI_DMA_TRANSMIT	SPI transmit data use DMA
SPI_DMA_RECEIVE	SPI receive data use DMA
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 transmit data DMA function */
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

spi_dma_disable

The description of spi_dma_disable is shown as below:

Table 3-735. Function spi_dma_disable

Function name	spi_dma_disable
Function prototype	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
Function descriptions	disable SPI DMA function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Input parameter{in}	
dma	SPI DMA mode
SPI_DMA_TRANSMIT	SPI transmit data use DMA
SPI_DMA_RECEIVE	SPI receive data use DMA
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 transmit data DMA function */
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

spi_i2s_data_frame_format_config

The description of spi_i2s_data_frame_format_config is shown as below:

Table 3-736. Function spi_i2s_data_frame_format_config

Function name	spi_i2s_data_frame_format_config
Function prototype	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
Function descriptions	configure SPI/I2S data frame format
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Input parameter{in}	
frame_format	SPI frame size
SPI_FRAMESIZE_16BIT	SPI frame size is 16 bits
SPI_FRAMESIZE_8BIT	SPI frame size is 8 bits
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure SPI1/I2S1 data frame format size is 16 bits */
spi_i2s_data_frame_format_config(SPI1, SPI_FRAMESIZE_16BIT);
```

spi_i2s_data_transmit

The description of spi_i2s_data_transmit is shown as below:

Table 3-737. Function spi_i2s_data_transmit

Function name	spi_i2s_data_transmit
Function prototype	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
Function descriptions	SPI transmit data
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Input parameter{in}	
data	16-bit data
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 transmit data */

uint16_t spi_send_array[] = {0x5050,0XA0A0};

spi_i2s_data_transmit(SPI0, spi0_send_array[0]);
```

spi_i2s_data_receive

The description of spi_i2s_data_receive is shown as below:

Table 3-738. Function spi_i2s_data_receive

Function name	spi_i2s_data_receive
Function prototype	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
Function descriptions	SPI receive data
Precondition	-
The called functions	-
Input parameter{in}	

spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
uint16_t	16-bit data

Example:

```
/* SPI0 receive data */

uint16_t spi0_receive_data;

spi0_receive_data = spi_i2s_data_receive(SPI0);
```

spi_bidirectional_transfer_config

The description of **spi_bidirectional_transfer_config** is shown as below:

Table 3-739. Function spi_bidirectional_transfer_config

Function name	spi_bidirectional_transfer_config
Function prototype	void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);
Function descriptions	configure SPI bidirectional transfer direction
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Input parameter{in}	
transfer_direction	SPI transfer direction
<i>SPI_BIDIRECTIONAL_TRANSMIT</i>	SPI work in transmit-only mode
<i>SPI_BIDIRECTIONAL_RECEIVE</i>	SPI work in receive-only mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 works in transmit-only mode */

spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

i2s_full_duplex_mode_config

The description of i2s_full_duplex_mode_config is shown as below:

Table 3-740. Function i2s_full_duplex_mode_config

Function name	i2s_full_duplex_mode_config
Function prototype	void i2s_full_duplex_mode_config (uint32_t i2s_add_periph,uint32_t i2s_mode,uint32_t i2s_standard,uint32_t i2s_ckpl,uint32_t frameformat);
Function descriptions	configure i2s full duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
<i>i2s_add_periph</i>	I2Sx_ADDperipheral
<i>I2Sx_ADD</i>	x=1,2
Input parameter{in}	
<i>i2s_mode</i>	i2s mode
<i>I2S_MODE_SLAVE_TX</i>	I2S slave transmit mode
<i>I2S_MODE_SLAVE_RX</i>	I2S slave receive mode
<i>I2S_MODE_MASTER_T</i> X	I2S master transmit mode
<i>I2S_MODE_MASTER_R</i> X	I2S master receive mode
Input parameter{in}	
<i>i2s_standard</i>	i2s standard
<i>I2S_STD_PHILLIPS</i>	I2S phillips standard
<i>I2S_STD_MSB</i>	I2S MSB standard
<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCM_SHORT</i>	I2S PCM short standard
<i>I2S_STD_PCM_LONG</i>	I2S PCM long standard
Input parameter{in}	
<i>i2s_ckpl</i>	i2s idle state clock polarity
<i>I2S_CKPL_LOW</i>	The idle state of I2S_CK is low level
<i>I2S_CKPL_HIGH</i>	The idle state of I2S_CK is high level
Input parameter{in}	
<i>i2s_frameformat</i>	i2s data length and channel length
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S data length is 16 bit and channel length is 16 bit
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S data length is 16 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S data length is 24 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S data length is 32 bit and channel length is 32 bit

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure I2S full duplex mode */

I2S_full_duplex_mode_config(I2S1_ADD,I2S_MODE_SLAVE_TX,I2S_STD_LSB,I2S_CKPL_
LOW,I2S_FRAMEFORMAT_DT16B_CH16B);
```

spi_i2s_format_error_clear

The description of spi_i2s_format_error_clear is shown as below:

Table 3-741. Function spi_i2s_format_error_clear

Function name	spi_i2s_format_error_clear
Function prototype	void spi_i2s_format_error_clear(uint32_t spi_periph, uint32_t flag);
Function descriptions	clear TI mode format error flag status
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Input parameter{in}	
flag	SPI/I2S frame format error flag
SPI_FLAG_FERR	SPI TI mode frame format error
I2S_FLAG_FERR	I2S frame format error
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear SPI0 TI mode format error flag */

spi_i2s_format_error_clear(SPI0, SPI_FLAG_FERR);
```

spi_crc_polynomial_set

The description of spi_crc_polynomial_set is shown as below:

Table 3-742. Function spi_crc_polynomial_set

Function name	spi_crc_polynomial_set
Function prototype	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);

Function descriptions	set SPI CRC polynomial
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Input parameter{in}	
crc_poly	CRC polynomial value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* set SPI0 CRC polynomial */

uint16_t CRC_VALUE = 0x5050;

spi_crc_polynomial_set(SPI0,CRC_VALUE);

```

spi_crc_polynomial_get

The description of spi_crc_polynomial_get is shown as below:

Table 3-743. Function spi_crc_polynomial_get

Function name	spi_crc_polynomial_get
Function prototype	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
Function descriptions	get SPI CRC polynomial
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
uint16_t	16-bit CRC polynomial (0-0xFFFF)

Example:

```

/* get SPI0 CRC polynomial */

uint16_t crc_val;

crc_val = spi_crc_polynomial_get(SPI0);

```

spi_crc_on

The description of spi_crc_on is shown as below:

Table 3-744. Function spi_crc_on

Function name	spi_crc_on
Function prototype	void spi_crc_on(uint32_t spi_periph);
Function descriptions	turn on SPI CRC function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn on SPI0 CRC function */
spi_crc_on(SPI0);
```

spi_crc_off

The description of spi_crc_off is shown as below:

Table 3-745. Function spi_crc_off

Function name	spi_crc_off
Function prototype	void spi_crc_off(uint32_t spi_periph);
Function descriptions	turn off SPI CRC function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn off SPI0 CRC function */
spi_crc_off(SPI0);
```

spi_crc_next

The description of spi_crc_next is shown as below:

Table 3-746. Function spi_crc_next

Function name	spi_crc_next
Function prototype	void spi_crc_next(uint32_t spi_periph);
Function descriptions	SPI next data is CRC value
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 next data is CRC value */
spi_crc_next(SPI0);
```

spi_crc_get

The description of spi_crc_get is shown as below:

Table 3-747. Function spi_crc_get

Function name	spi_crc_get
Function prototype	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t spi_crc);
Function descriptions	get SPI CRC send value or receive value
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Input parameter{in}	
spi_crc	SPI crc value
SPI_CRC_TX	get transmit crc value
SPI_CRC_RX	get receive crc value
Output parameter{out}	
-	-
Return value	
uint16_t	16-bit CRC value (0-0xFFFF)

Example:

```
/* get SPI0 CRC send value */

uint16_t crc_val;

crc_val = spi_crc_get(SPI0, SPI_CRC_TX);
```

spi_crc_error_clear

The description of **spi_crc_error_clear** is shown as below:

Table 3-748. Function `spi_crc_error_clear`

Function name	spi_crc_error_clear
Function prototype	void spi_crc_error_clear(uint32_t spi_periph);
Function descriptions	clear SPI CRC error flag status
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear SPI0 CRC error flag status */

spi_crc_error_clear(SPI0);
```

spi_ti_mode_enable

The description of **spi_ti_mode_enable** is shown as below:

Table 3-749. Function `spi_ti_mode_enable`

Function name	spi_ti_mode_enable
Function prototype	void spi_ti_mode_enable(uint32_t spi_periph);
Function descriptions	enable SPI TI mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* enable SPI0 TI mode */
spi_ti_mode_enable(SPI0);
```

spi_ti_mode_disable

The description of spi_ti_mode_disable is shown as below:

Table 3-750. Function spi_ti_mode_disable

Function name	spi_ti_mode_disable
Function prototype	void spi_ti_mode_disable(uint32_t spi_periph);
Function descriptions	disable SPI TI mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

spi_quad_enable

The description of spi_quad_enable is shown as below:

Table 3-751. Function spi_quad_enable

Function name	spi_quad_enable
Function prototype	void spi_quad_enable(uint32_t spi_periph);
Function descriptions	enable quad wire SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=5
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable SPI5 quad wire mode */
spi_quad_enable(SPI5);
```

spi_quad_disable

The description of spi_quad_disable is shown as below:

Table 3-752. Function spi_quad_disable

Function name	spi_quad_disable
Function prototype	spi_quad_disable(uint32_t spi_periph);
Function descriptions	disable quad wire SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI5 quad wire mode */
spi_quad_disable(SPI5);
```

spi_quad_write_enable

The description of spi_quad_write_enable is shown as below:

Table 3-753. Function spi_quad_write_enable

Function name	spi_quad_write_enable
Function prototype	void spi_quad_write_enable(uint32_t spi_periph);
Function descriptions	enable quad wire SPI write
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=5

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI5 quad wire write */

spi_quad_write_enable(SPI5);
```

spi_quad_read_enable

The description of spi_quad_read_enable is shown as below:

Table 3-754. Function spi_quad_read_enable

Function name	spi_quad_read_enable
Function prototype	void spi_quad_read_enable(uint32_t spi_periph);
Function descriptions	enable quad wire SPI read
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI5 quad wire read */

spi_quad_read_enable(SPI5);
```

spi_quad_io23_output_enable

The description of spi_quad_io23_output_enable is shown as below:

Table 3-755. Function spi_quad_io23_output_enable

Function name	spi_quad_io23_output_enable
Function prototype	void spi_quad_io23_output_enable(uint32_t spi_periph);
Function descriptions	enable SPI_IO2 and SPI_IO3 pin output
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral

SPIx	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI5 SPI_IO2 and SPI_IO3 pin output */
spi_quad_io23_output_enable(SPI5);
```

spi_quad_io23_output_disable

The description of spi_quad_io23_output_disable is shown as below:

Table 3-756. Function spi_quad_io23_output_disable

Function name	spi_quad_io23_output_disable
Function prototype	void spi_quad_io23_output_disable(uint32_t spi_periph);
Function descriptions	disable SPI_IO2 and SPI_IO3 pin output
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI5 SPI_IO2 and SPI_IO3 pin output */
spi_quad_io23_output_disable(SPI5);
```

spi_i2s_flag_get

The description of spi_i2s_flag_get is shown as below:

Table 3-757. Function spi_i2s_flag_get

Function name	spi_i2s_flag_get
Function prototype	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t spi_i2s_flag);
Function descriptions	get SPI and I2S flag status
Precondition	-
The called functions	-
Input parameter{in}	

spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Input parameter{in}	
spi_i2s_flag	SPI/I2S flag status
<i>SPI_FLAG_TBE</i>	transmit buffer empty flag
<i>SPI_FLAG_RBNE</i>	receive buffer not empty flag
<i>SPI_FLAG_TRANS</i>	transmit on-going flag
<i>SPI_FLAG_RXORERR</i>	receive overrun error flag
<i>SPI_FLAG_CONFERR</i>	mode config error flag
<i>SPI_FLAG_CRCERR</i>	CRC error flag
<i>SPI_FLAG_FERR</i>	format error flag
<i>I2S_FLAG_TBE</i>	transmit buffer empty flag
<i>I2S_FLAG_RBNE</i>	receive buffer not empty flag
<i>I2S_FLAG_TRANS</i>	transmit on-going flag
<i>I2S_FLAG_RXORERR</i>	overrun error flag
<i>I2S_FLAG_TXURERR</i>	underrun error flag
<i>I2S_FLAG_CH</i>	channel side flag
<i>I2S_FLAG_FERR</i>	format error flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty flag status */

while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));

spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

spi_i2s_interrupt_enable

The description of **spi_i2s_interrupt_enable** is shown as below:

Table 3-758. Function **spi_i2s_interrupt_enable**

Function name	spi_i2s_interrupt_enable
Function prototype	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t spi_i2s_int);
Function descriptions	enable SPI and I2S interrupt
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Input parameter{in}	

spi_i2s_int	SPI/I2S interrupt
<i>SPI_I2S_INT_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 transmit buffer empty interrupt */
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

spi_i2s_interrupt_disable

The description of spi_i2s_interrupt_disable is shown as below:

Table 3-759. Function spi_i2s_interrupt_disable

Function name	spi_i2s_interrupt_disable
Function prototype	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t spi_i2s_int);
Function descriptions	disable SPI and I2S interrupt
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Input parameter{in}	
spi_i2s_int	SPI/I2S interrupt
<i>SPI_I2S_INT_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 transmit buffer empty interrupt */
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

spi_i2s_interrupt_flag_get

The description of spi_i2s_interrupt_flag_get is shown as below:

Table 3-760. Function spi_i2s_interrupt_flag_get

Function name	spi_i2s_interrupt_flag_get
Function prototype	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t spi_i2s_int);
Function descriptions	get SPI and I2S interrupt status
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Input parameter{in}	
spi_i2s_int	SPI/I2S interrupt flag status
SPI_I2S_INT_FLAG_T BE	transmit buffer empty interrupt
SPI_I2S_INT_FLAG_R BNE	receive buffer not empty interrupt
SPI_I2S_INT_FLAG_R XORERR	overrun interrupt
SPI_INT_FLAG_CONF ERR	config error interrupt
SPI_INT_FLAG_CRCE RR	CRC error interrupt
I2S_INT_FLAG_TXUR ERR	underrun error interrupt
I2S_INT_FLAG_FERR	format error interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty interrupt status */
if(RESET != spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE)){
    while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
    spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
}
```

3.25. SYSCFG

The SYSCFG registers are listed in chapter [3.25.1](#), the TIMER firmware functions are introduced in chapter [3.25.2](#).

3.25.1. Descriptions of Peripheral registers

SYSCFG registers are listed in the table shown as below:

Table 3-761. SYSCFG registers

Registers	Descriptions
SYSCFG_CFG0	Configuration register 0
SYSCFG_CFG1	Configuration register 1
SYSCFG_EXTISSL0	EXTI sources selection register 0
SYSCFG_EXTISSL1	EXTI sources selection register 1
SYSCFG_EXTISSL2	EXTI sources selection register 2
SYSCFG_EXTISSL3	EXTI sources selection register 3
SYSCFG_CPSCTL	I/O compensation control register

3.25.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

Table 3-762. SYSCFG firmware function

Function name	Function description
syscfg_deinit	deinit syscfg module
syscfg_bootmode_config	configure the boot mode
syscfg_fmc_swap_config	configure FMC memory mapping swap
syscfg_exmc_swap_config	configure the EXMC swap
syscfg_exti_line_config	configure the GPIO pin as EXTI Line
syscfg_enet_phy_interface_config	configure the PHY interface for the ethernet MAC
syscfg_compensation_config	configure the I/O compensation cell
syscfg_flag_get	check the I/O compensation cell is ready or not

syscfg_deinit

The description of syscfg_deinit is shown as below:

Table 3-763. Function syscfg_deinit

Function name	syscfg_deinit
Function prototype	void syscfg_deinit(void);
Function descriptions	deinit syscfg module
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset SYSCFG */
syscfg_deinit();
```

syscfg_bootmode_config

The description of syscfg_bootmode_config is shown as below:

Table 3-764. Function syscfg_bootmode_config

Function name	syscfg_bootmode_config
Function prototype	void syscfg_bootmode_config(uint8_t syscfg_bootmode);
Function descriptions	configure the boot mode
Precondition	-
The called functions	-
Input parameter{in}	
syscfg_bootmode	selects the memory remapping
SYSCFG_BOOTMODE	main flash memory (0x08000000~0x083BFFFF) is mapped at address

<i>_FLASH</i>	0x00000000
<i>SYSCFG_BOOTMODE_BOOTLOADER</i>	boot loader (0x1FFF0000 - 0x1FFF77FF) is mapped at address 0x00000000
<i>SYSCFG_BOOTMODE_EXMC_SRAM</i>	SRAM/NOR 0 and 1 of EXMC (0x60000000~0x67FFFFFF) is mapped at address 0x00000000
<i>SYSCFG_BOOTMODE_SRAM</i>	SRAM0 of on-chip SRAM (0x20000000~0x2001BFFF) is mapped at address 0x00000000
<i>SYSCFG_BOOTMODE_EXMC_SDRAM</i>	SDRAM bank0 of EXMC (0xC0000000~0xC7FFFFFF) is mapped at address 0x00000000
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure boot from main flash */
syscfg_bootmode_config(SYSCFG_BOOTMODE_FLASH);
```

syscfg_fmc_swap_config

The description of `syscfg_fmc_swap_config` is shown as below:

Table 3-765. Function `syscfg_fmc_swap_config`

Function name	syscfg_fmc_swap_config
Function prototype	void syscfg_fmc_swap_config(uint32_t syscfg_fmc_swap);
Function descriptions	FMC memory mapping swap
Precondition	-
The called functions	-
Input parameter{in}	
syscfg_fmc_swap	selects the interal flash bank swapping
<i>SYSCFG_FMC_SWP_BANK0</i>	bank 0 is mapped at address 0x08000000 and bank 1 is mapped at address 0x08100000
<i>SYSCFG_FMC_SWP_BANK1</i>	bank 1 is mapped at address 0x08000000 and bank 0 is mapped at address 0x08100000

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* FMC memory bank 0 is mapped at address 0x08000000 and bank 1 is mapped at address
0x08100000 */

syscfg_fmc_swap_config(SYSCFG_FMC_SWP_BANK0);
```

syscfg_exmc_swap_config

The description of syscfg_exmc_swap_config is shown as below:

Table 3-766. Function syscfg_exmc_swap_config

Function name	syscfg_exmc_swap_config
Function prototype	void syscfg_exmc_swap_config(uint32_t syscfg_exmc_swap);
Function descriptions	EXMC memory mapping swap
Precondition	-
The called functions	-
Input parameter{in}	
syscfg_exmc_swap	selects the memories in EXMC swapping
SYSCFG_EXMC_SWP_ENABLE	SDRAM bank 0 and bank 1 are swapped with NAND bank 1 and PC card
SYSCFG_EXMC_SWP_DISABLE	no memory mapping swap
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable EXMC memory mapping swap */

syscfg_exmc_swap_config(SYSCFG_EXMC_SWP_ENABLE);
```

syscfg_exti_line_config

The description of syscfg_exti_line_config is shown as below:

Table 3-767. Function syscfg_exti_line_config

Function name	syscfg_exti_line_config
Function prototype	void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);
Function descriptions	configure the GPIO pin as EXTI Line
Precondition	-
The called functions	-
Input parameter{in}	
exti_port	specify the GPIO port used in EXTI
<i>EXTI_SOURCE_GPIOx</i>	x = A,B,C,D,E,F,G,H,I
Input parameter{in}	
exti_pin	specify the EXTI line
<i>EXTI_SOURCE_PINx</i>	x = 0..15
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PA0 pin as EXTI Line */
syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN0);
```

syscfg_enet_phy_interface_config

The description of syscfg_enet_phy_interface_config is shown as below:

Table 3-768. Function syscfg_enet_phy_interface_config

Function name	syscfg_enet_phy_interface_config
Function prototype	void syscfg_enet_phy_interface_config(uint32_t syscfg_enet_phy_interface);
Function descriptions	configure the PHY interface for the ethernet MAC

Precondition	-
The called functions	-
Input parameter{in}	
syscfg_enet_phy_interface	specifies the media interface mode
SYSCFG_ENET_PHY_MII	MII mode is selected
SYSCFG_ENET_PHY_RMII	RMII mode is selected
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the MII PHY interface for the ethernet MAC */
syscfg_enet_phy_interface_config(SYSCFG_ENET_PHY_MII);
```

syscfg_compensation_config

The description of syscfg_compensation_config is shown as below:

Table 3-769. Function syscfg_compensation_config

Function name	syscfg_compensation_config
Function prototype	void syscfg_compensation_config(uint32_t syscfg_compensation);
Function descriptions	configure the I/O compensation cell
Precondition	-
The called functions	-
Input parameter{in}	
syscfg_compensation	specifies the I/O compensation cell mode
SYSCFG_COMPENSATION_ENABLE	I/O compensation cell is enabled
SYSCFG_COMPENSATION_DISABLE	I/O compensation cell is disabled

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the I/O compensation cell function */

syscfg_compensation_config(SYSCFG_COMPENSATION_ENABLE);
```

syscfg_flag_get

The description of syscfg_flag_get is shown as below:

Table 3-770. Function syscfg_flag_get

Function name	syscfg_flag_get
Function prototype	FlagStatus syscfg_flag_get(void);
Function descriptions	checks whether the I/O compensation cell ready flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check whether the I/O compensation cell ready flag is set or not */

if(RESET != syscfg_flag_get());
```

3.26. TIMER

The timers have a 16-bit counter that can be used as an unsigned counter and supports both input capture and output compare. Timers (TIMERx) are divided into five sorts: advanced timer (TIMERx, x=0, 7), general level0 timer (TIMERx, x=1, 2, 3, 4), general level1 timer

(TIMERx, x=8, 11), general level2 timer (TIMERx, x=9, 10, 12, 13), Basic timer (TIMERx, x=5, 6). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.26.1](#), the TIMER firmware functions are introduced in chapter [3.26.2](#).

3.26.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

Table 3-771. TIMERx Registers

Registers	Descriptions
TIMER_CTL0	Control register 0
TIMER_CTL1	Control register 1
TIMER_SMCFG	Slave mode configuration register
TIMER_DMINTEN	DMA and interrupt enable register
TIMER_INTF	Interrupt flag register
TIMER_SWEVG	Software event generation register
TIMER_CHCTL0	Channel control register 0
TIMER_CHCTL1	Channel control register 1
TIMER_CHCTL2	Channel control register 2
TIMER_CNT	Counter register
TIMER_PSC	Prescaler register
TIMER_CAR	Counter auto reload register
TIMER_CREP	Counter repetition register
TIMER_CH0CV	Channel 0 capture/compare value register
TIMER_CH1CV	Channel 1 capture/compare value register
TIMER_CH2CV	Channel 2 capture/compare value register
TIMER_CH3CV	Channel 3 capture/compare value register
TIMER_CCHP	Channel complementary protection register
TIMER_DMACFG	DMA configuration register
TIMER_DMATB	DMA transfer buffer register
TIMER_IRMP	TIMER input remap register
TIMER_CFG	Configuration register

3.26.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

Table 3-772.TIMERx firmware function

Function name	Function description
timer_deinit	deinit a timer
timer_struct_para_init	initialize the parameters of TIMER init parameter struct with the default values
timer_init	initialize TIMER counter
timer_enable	enable a timer
timer_disable	disable a timer
timer_auto_reload_shadow_enable	enable the auto reload shadow function
timer_auto_reload_shadow_disable	disable the auto reload shadow function
timer_update_event_enable	enable the update event
timer_update_event_disable	disable the update event
timer_counter_alignment	set TIMER counter alignment mode
timer_counter_up_direction	set TIMER counter up direction
timer_counter_down_direction	set TIMER counter down direction
timer_prescaler_config	configure TIMER prescaler
timer_repetition_value_config	configure TIMER repetition register value
timer_autoreload_value_config	configure TIMER autoreload register value
timer_counter_value_config	configure TIMER counter register value
timer_counter_read	read TIMER counter value
timer_prescaler_read	read TIMER prescaler value
timer_single_pulse_mode_config	configure TIMER single pulse mode
timer_update_source_config	configure TIMER update source
timer_dma_enable	enable the TIMER DMA
timer_dma_disable	disable the TIMER DMA
timer_channel_dma_request_source_select	channel DMA request source selection

Function name	Function description
timer_dma_transfer_config	configure the TIMER DMA transfer
timer_event_software_generate	software generate events
timer_break_struct_para_init	initialize the parameters of TIMER break parameter struct with the default values
timer_break_config	configure TIMER break function
timer_break_enable	enable TIMER break function
timer_break_disable	disable TIMER break function
timer_automatic_output_enable	enable TIMER output automatic function
timer_automatic_output_disable	disable TIMER output automatic function
timer_primary_output_config	configure TIMER primary output function
timer_channel_control_shadow_config	channel capture/compare control shadow register enable
timer_channel_control_shadow_update_config	configure TIMER channel control shadow register update control
timer_channel_output_struct_para_init	initialize the parameters of TIMER channel output parameter struct with the default values
timer_channel_output_config	configure TIMER channel output function
timer_channel_output_mode_config	configure TIMER channel output compare mode
timer_channel_output_pulse_value_config	configure TIMER channel output pulse value
timer_channel_output_shadow_config	configure TIMER channel output shadow function
timer_channel_output_fast_config	configure TIMER channel output fast function
timer_channel_output_clear_config	configure TIMER channel output clear function
timer_channel_output_polarity_config	configure TIMER channel output polarity
timer_channel_complementary_output_polarity_config	configure TIMER channel complementary output polarity
timer_channel_output_state_config	configure TIMER channel enable state
timer_channel_complementary_output_enable_config	configure TIMER channel complementary output enable state

Function name	Function description
output_state_config	
timer_channel_input_struct_para_init	initialize the parameters of TIMER channel input parameter struct with the default values
timer_input_capture_config	configure TIMER input capture parameter
timer_channel_input_capture_prescaler_config	configure TIMER channel input capture prescaler value
timer_channel_capture_value_register_read	read TIMER channel capture compare register value
timer_input_pwm_capture_config	configure TIMER input pwm capture function
timer_hall_mode_config	configure TIMER hall sensor mode
timer_input_trigger_source_select	select TIMER input trigger source
timer_master_output_trigger_source_select	select TIMER master mode output trigger source
timer_slave_mode_select	select TIMER slave mode
timer_master_slave_mode_config	configure TIMER master slave mode
timer_external_trigger_config	configure TIMER external trigger input
timer_quadrature_decoder_mode_config	configure TIMER quadrature decoder mode
timer_internal_clock_config	configure TIMER internal clock mode
timer_internal_trigger_as_external_clock_config	configure TIMER the internal trigger as external clock input
timer_external_trigger_as_external_clock_config	configure TIMER the external trigger as external clock input
timer_external_clock_mode0_config	configure TIMER the external clock mode 0
timer_external_clock_mode1_config	configure TIMER the external clock mode 1
timer_external_clock_mode1_disable	disable TIMER the external clock mode 1
timer_write_chxval_register_config	configure TIMER write CHxVAL register selection
timer_output_value_selection_config	configure TIMER output value selection
timer_channel_remap_config	configure TIMER channel remap function

Function name	Function description
timer_flag_get	get TIMER flags
timer_flag_clear	clear TIMER flags
timer_interrupt_enable	enable the TIMER interrupt
timer_interrupt_disable	disable the TIMER interrupt
timer_interrupt_flag_get	get timer interrupt flag
timer_interrupt_flag_clear	clear TIMER interrupt flag

Structure timer_parameter_struct

Table 3-773. Structure timer_parameter_struct

Member name	Function description
prescaler	prescaler value (0~65535)
alignedmode	aligned mode (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)
counterdirection	counter direction (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)
period	period value (0~65535)
clockdivision	clock division value (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)
repetitioncounter	the counter repetition value (0~255)

Structure timer_break_parameter_struct

Table 3-774. Structure timer_break_parameter_struct

Member name	Function description
runoffstate	run mode off-state (TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE)
ideloffstate	idle mode off-state (TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE)
deadtime	dead time (0~255)
breakpolarity	break polarity (TIMER_BREAK_POLARITY_LOW, TIMER_BREAK_POLARITY_HIGH)
outputautostate	output automatic enable (TIMER_OUTAUTO_ENABLE, TIMER_OUTAUTO_DISABLE)

Member name	Function description
protectmode	complementary register protect control (TIMER_CCHP_PROT_OFF, TIMER_CCHP_PROT_0, TIMER_CCHP_PROT_1, TIMER_CCHP_PROT_2)
breakstate	break enable (TIMER_BREAK_ENABLE, TIMER_BREAK_DISABLE)

Structure timer_oc_parameter_struct

Table 3-775. Structure timer_oc_parameter_struct

Member name	Function description
outputstate	channel output state (TIMER_CCX_ENABLE, TIMER_CCX_DISABLE)
outputnstate	channel complementary output state (TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE)
ocpolarity	channel output polarity (TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW)
ocnpolarity	channel complementary output polarity (TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW)
ocidlestate	idle state of channel output (TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH)
ocnidlestate	idle state of channel complementary output (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH)

Structure timer_ic_parameter_struct

Table 3-776. Structure timer_ic_parameter_struct

Member name	Function description
icpolarity	channel input polarity (TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)
icselection	channel input mode selection (TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS)
icprescaler	channel input capture prescaler (TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)
icfilter	channel input capture filter control (0~15)

timer_deinit

The description of timer_deinit is shown as below:

Table 3-777. Function timer_deinit

Function name	timer_deinit
Function prototype	void timer_deinit(uint32_t timer_periph);
Function descriptions	deinit a TIMER
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset TIMER0 */
timer_deinit (TIMER0);
```

timer_struct_para_init

The description of timer_struct_para_init is shown as below:

Table 3-778. Function timer_struct_para_init

Function name	timer_struct_para_init
Function prototype	void timer_struct_para_init(timer_parameter_struct* initpara);
Function descriptions	initialize the parameters of TIMER init parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
initpara	TIMER init parameter struct, the structure members can refer to Table 3-773. Structure timer_parameter_struct .
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* initialize TIMER init parameter struct with a default value */

timer_parameter_struct timer_initpara;

timer_struct_para_init(timer_initpara);
```

timer_init

The description of timer_init is shown as below:

Table 3-779. Function timer_init

Function name	timer_init
Function prototype	void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);
Function descriptions	initialize TIMER counter
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Input parameter{in}	
initpara	TIMER init parameter struct, the structure members can refer to Table 3-773. Structure timer_parameter_struct.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER0 */

timer_parameter_struct timer_initpara;

timer_initpara.prescaler = 107;
```

```

timer_initpara.alignedmode      = TIMER_COUNTER_EDGE;
timer_initpara.counterdirection = TIMER_COUNTER_UP;
timer_initpara.period          = 999;
timer_initpara.clockdivision   = TIMER_CKDIV_DIV1;
timer_initpara.repetitioncounter = 1;
timer_init(TIMER0,&timer_initpara);

```

timer_enable

The description of timer_enable is shown as below:

Table 3-780. Function timer_enable

Function name	timer_enable
Function prototype	void timer_enable(uint32_t timer_periph);
Function descriptions	enable a timer
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable TIMER0 */

timer_enable (TIMER0);

```

timer_disable

The description of timer_disable is shown as below:

Table 3-781. Function timer_disable

Function name	timer_disable
----------------------	---------------

Function prototype	void timer_disable(uint32_t timer_periph);
Function descriptions	disable a timer
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 */

timer_disable (TIMER0);
```

timer_auto_reload_shadow_enable

The description of **timer_auto_reload_shadow_enable** is shown as below:

Table 3-782. Function timer_auto_reload_shadow_enable

Function name	timer_auto_reload_shadow_enable
Function prototype	void timer_auto_reload_shadow_enable(uint32_t timer_periph);
Function descriptions	enable the auto reload shadow function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable the TIMER0 auto reload shadow function */
timer_auto_reload_shadow_enable (TIMER0);
```

timer_auto_reload_shadow_disable

The description of timer_auto_reload_shadow_disable is shown as below:

Table 3-783. Function timer_auto_reload_shadow_disable

Function name	timer_auto_reload_shadow_disable
Function prototype	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
Function descriptions	disable the auto reload shadow function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..13)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 auto reload shadow function */
timer_auto_reload_shadow_disable (TIMER0);
```

timer_update_event_enable

The description of timer_update_event_enable is shown as below:

Table 3-784. Function timer_update_event_enable

Function name	timer_update_event_enable
Function prototype	void timer_update_event_enable(uint32_t timer_periph);
Function descriptions	enable the update event

Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER0 the update event */
timer_update_event_enable (TIMER0);
```

timer_update_event_disable

The description of **timer_update_event_disable** is shown as below:

Table 3-785. Function timer_update_event_disable

Function name	timer_update_event_disable
Function prototype	void timer_update_event_disable (uint32_t timer_periph);
Function descriptions	disable the update event
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 the update event */
```

```
timer_update_event_disable (TIMER0);
```

timer_counter_alignment

The description of timer_counter_alignment is shown as below:

Table 3-786. Function timer_counter_alignment

Function name	timer_counter_alignment
Function prototype	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);
Function descriptions	set TIMER counter alignment mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7..13)</i>	TIMER peripheral selection
Input parameter{in}	
aligned	alignment mode
<i>TIMER_COUNTER_EDGE</i>	No center-aligned mode (edge-aligned mode). The direction of the counter is specified by the DIR bit.
<i>TIMER_COUNTER_CENTER_DOWN</i>	Center-aligned and counting down assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0register). Only when the counter is counting down, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_CENTER_UP</i>	Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0register). Only when the counter is counting up, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_CENTER_BOTH</i>	Center-aligned and counting up/down assert mode. The counter counts under center-aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0 register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set.
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* set TIMER0 counter center-aligned and counting up assert mode */
timer_counter_alignment(TIMER0, TIMER_COUNTER_CENTER_UP);
```

timer_counter_up_direction

The description of timer_counter_up_direction is shown as below:

Table 3-787. Function timer_counter_up_direction

Function name	timer_counter_up_direction
Function prototype	void timer_counter_up_direction(uint32_t timer_periph);
Function descriptions	set TIMER counter up direction
Precondition	set TIMER counter no center-aligned mode (edge-aligned mode)
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set TIMER0 counter up direction */
timer_counter_up_direction(TIMER0);
```

timer_counter_down_direction

The description of timer_counter_down_direction is shown as below:

Table 3-788. Function timer_counter_down_direction

Function name	timer_counter_down_direction
Function prototype	void timer_counter_down_direction(uint32_t timer_periph);
Function descriptions	set TIMER counter down direction

Precondition	set TIMER counter no center-aligned mode (edge-aligned mode)
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set TIMER0 counter down direction */
timer_counter_down_direction (TIMER0);
```

timer_prescaler_config

The description of timer_prescaler_config is shown as below:

Table 3-789. Function timer_prescaler_config

Function name	timer_prescaler_config
Function prototype	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t pscreload);
Function descriptions	configure TIMER prescaler
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Input parameter{in}	
prescaler	prescaler value (0~65535)
Input parameter{in}	
pscreload	prescaler reload mode

<i>TIMER_PSC_RELOAD_NOW</i>	the prescaler is loaded right now
<i>TIMER_PSC_RELOAD_UPDATE</i>	the prescaler is loaded at the next update event
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 prescaler */
timer_prescaler_config (TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

timer_repetition_value_config

The description of timer_repetition_value_config is shown as below:

Table 3-790. Function timer_repetition_value_config

Function name	timer_repetition_value_config
Function prototype	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
Function descriptions	configure TIMER repetition register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
repetition	the counter repetition value (0~255)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 repetition register value */

timer_repetition_value_config (TIMER0, 98);

```

timer_autoreload_value_config

The description of timer_autoreload_value_config is shown as below:

Table 3-791. Function timer_autoreload_value_config

Function name	timer_autoreload_value_config
Function prototype	void timer_autoreload_value_config(uint32_t timer_periph, uint32_t autoreload);
Function descriptions	configure TIMER autoreload register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Input parameter{in}	
autoreload	the counter auto-reload value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER autoreload register value */

timer_autoreload_value_config (TIMER0, 3000);

```

timer_counter_value_config

The description of timer_counter_value_config is shown as below:

Table 3-792. Function timer_counter_value_config

Function name	timer_counter_value_config
----------------------	----------------------------

Function prototype	void timer_counter_value_config(uint32_t timer_periph, uint32_t counter);
Function descriptions	configure TIMER counter register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Input parameter{in}	
counter	the counter value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 counter register value */

timer_counter_value_config (TIMER0);
```

timer_counter_read

The description of timer_counter_read is shown as below:

Table 3-793. Function timer_counter_read

Function name	timer_counter_read
Function prototype	uint32_t timer_counter_read(uint32_t timer_periph);
Function descriptions	read TIMER counter value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	

-	-
Return value	
uint32_t	counter value

Example:

```
/* read TIMER0 counter value */

uint32_t i = 0;

i = timer_counter_read (TIMER0);
```

timer_prescaler_read

The description of **timer_prescaler_read** is shown as below:

Table 3-794. Function timer_prescaler_read

Function name	timer_prescaler_read
Function prototype	uint16_t timer_prescaler_read(uint32_t timer_periph);
Function descriptions	read TIMER prescaler value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
uint16_t	prescaler register value (0x0000~0xFFFF)

Example:

```
/* read TIMER0 prescaler value */

uint16_t i = 0;

i = timer_prescaler_read (TIMER0);
```

timer_single_pulse_mode_config

The description of **timer_single_pulse_mode_config** is shown as below:

Table 3-795. Function timer_single_pulse_mode_config

Function name	timer_single_pulse_mode_config
Function prototype	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
Function descriptions	configure TIMER single pulse mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..8,11)</i>	TIMER peripheral selection
Input parameter{in}	
spmode	pulse mode
<i>TIMER_SP_MODE_S/N GLE</i>	single pulse mode
<i>TIMER_SP_MODE_R/E PETITIVE</i>	repetitive pulse mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 single pulse mode */

timer_single_pulse_mode_config (TIMER0, TIMER_SP_MODE_SINGLE);

```

timer_update_source_config

The description of timer_update_source_config is shown as below:

Table 3-796. Function timer_update_source_config

Function name	timer_update_source_config
Function prototype	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
Function descriptions	configure TIMER update source

Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Input parameter{in}	
update	update source
<i>TIMER_UPDATE_SRC_GLOBAL</i>	Any of the following events generate an update interrupt or DMA request: - The UPG bit is set - The counter generates an overflow or underflow event - The slave mode controller generates an update event
<i>TIMER_UPDATE_SRC_REGULAR</i>	Only counter overflow/underflow generates an update interrupt or DMA request.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER update only by counter overflow/underflow */
timer_update_source_config (TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

timer_dma_enable

The description of timer_dma_enable is shown as below:

Table 3-797. Function timer_dma_enable

Function name	timer_dma_enable
Function prototype	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
Function descriptions	enable the TIMER DMA
Precondition	-
The called functions	-
Input parameter{in}	

timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
dma	timer DMA source enable
<i>TIMER_DMA_UPD</i>	update DMA enable, TIMERx(x=0..7)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable, TIMERx(x=0,7)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable, TIMERx(x=0..4,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TIMER0 update DMA */
timer_dma_enable (TIMER0, TIMER_DMA_UPD);
```

timer_dma_disable

The description of timer_dma_disable is shown as below:

Table 3-798. Function timer_dma_disable

Function name	timer_dma_disable
Function prototype	void timer_dma_disable (uint32_t timer_periph, uint16_t dma);
Function descriptions	disable the TIMER DMA
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral

<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
dma	timer DMA source disable
<i>TIMER_DMA_UPD</i>	update DMA disable, TIMERx(x=0..7)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA disable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA disable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA disable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA disable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CMTD</i>	commutation DMA request disable, TIMERx(x=0,7)
<i>TIMER_DMA_TRGD</i>	trigger DMA disable, TIMERx(x=0..4,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 update DMA */
timer_dma_disable (TIMER0, TIMER_DMA_UPD);
```

timer_channel_dma_request_source_select

The description of `timer_channel_dma_request_source_select` is shown as below:

Table 3-799. Function `timer_channel_dma_request_source_select`

Function name	timer_channel_dma_request_source_select
Function prototype	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
Function descriptions	channel DMA request source selection
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral

<i>TIMERx</i>	TIMER peripheral selection
Input parameter{in}	
dma_request	channel DMA request source selection
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	DMA request of channel n is sent when channel y event occurs
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	DMA request of channel n is sent when update event occurs
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* TIMER0 channel DMA request of channel n is sent when channel y event occurs */

timer_channel_dma_request_source_select(TIMER0,
TIMER_DMAREQUEST_CHANNELEVENT);
```

timer_dma_transfer_config

The description of timer_dma_transfer_config is shown as below:

Table 3-800. Function timer_dma_transfer_config

Function name	timer_dma_transfer_config
Function prototype	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
Function descriptions	configure the TIMER DMA transfer
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
dma_baseaddr	DMA transfer access start address

<i>TIMER_DMACFG_DMA_TA_CTL0</i>	DMA transfer address is TIMER_CTL0, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CTL1</i>	DMA transfer address is TIMER_CTL1, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_SMCFG</i>	DMA transfer address is TIMER_SMCFG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_DMAINTEN</i>	DMA transfer address is TIMER_DMAINTEN, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_INTF</i>	DMA transfer address is TIMER_INTF, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_SWEVG</i>	DMA transfer address is TIMER_SWEVG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CHCTL0</i>	DMA transfer address is TIMER_CHCTL0, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CHCTL1</i>	DMA transfer address is TIMER_CHCTL1, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CHCTL2</i>	DMA transfer address is TIMER_CHCTL2, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CNT</i>	DMA transfer address is TIMER_CNT, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_PSC</i>	DMA transfer address is TIMER_PSC, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CAR</i>	MA transfer address is TIMER_CAR, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CREP</i>	DMA transfer address is TIMER_CREP, TIMERx(x=0..7)
<i>TIMER_DMACFG_DMA_TA_CH0CV</i>	DMA transfer address is TIMER_CH0CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CH1CV</i>	DMA transfer address is TIMER_CH1CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CH2CV</i>	DMA transfer address is TIMER_CH2CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CH3CV</i>	DMA transfer address is TIMER_CH3CV, TIMERx(x=0..4,7)

<i>TIMER_DMACFG_DMA_TA_CCHP</i>	DMA transfer address is TIMER_CCHP, TIMERx(x=0..7)
<i>TIMER_DMACFG_DMA_TA_DMACFG</i>	DMA transfer address is TIMER_DMACFG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_DMATB</i>	DMA transfer address is TIMER_DMATB, TIMERx(x=0..4,7)
Input parameter{in}	
dma_lenth	DMA transfer count
<i>TIMER_DMACFG_DMA_TC_xTRANSFER</i>	x=1..18, DMA transfer x time
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the TIMER0 DMA transfer */
timer_dma_transfer_config(TIMER0, TIMER_DMACFG_DMATA_CTL0,
    TIMER_DMACFG_DMATC_5TRANSFER);
```

timer_event_software_generate

The description of timer_event_software_generate is shown as below:

Table 3-801. Function timer_event_software_generate

Function name	timer_event_software_generate
Function prototype	void timer_event_software_generate(uint32_t timer_periph, uint16_t event);
Function descriptions	software generate events
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters

Input parameter{in}	
event	the timer software event generation sources
<i>TIMER_EVENT_SRC_UPG</i>	update event,TIMERx(x=0..13)
<i>TIMER_EVENT_SRC_CH0G</i>	channel 0 capture or compare event generation,TIMERx(x=0..4,7..13)
<i>TIMER_EVENT_SRC_CH1G</i>	channel 1 capture or compare event generation,TIMERx(x=0..4,7,8,11)
<i>TIMER_EVENT_SRC_CH2G</i>	channel 2 capture or compare event generation,TIMERx(x=0..4,7)
<i>TIMER_EVENT_SRC_CH3G</i>	channel 3 capture or compare event generation,TIMERx(x=0..4,7)
<i>TIMER_EVENT_SRC_CMTG</i>	channel commutation event generation,TIMERx(x=0,7)
<i>TIMER_EVENT_SRC_TRIGGER</i>	trigger event generation,TIMERx(x=0..4,7,8,11)
<i>TIMER_EVENT_SRC_BREAK</i>	break event generation,TIMERx(x=0,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* software generate update event*/
timer_event_software_generate (TIMER0, TIMER_EVENT_SRC_UPG);

```

timer_break_struct_para_init

The description of `timer_break_struct_para_init` is shown as below:

Table 3-802. Function `timer_break_struct_para_init`

Function name	timer_break_struct_para_init
Function prototype	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);

Function descriptions	initialize the parameters of TIMER break parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
breakpara	TIMER break parameter struct, the structure members can refer to Table 3-774, Structure timer break parameter struct.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER break parameter struct with a default value */

timer_break_parameter_struct timer_breakpara;

timer_break_struct_para_init(timer_breakpara);
```

timer_break_config

The description of timer_break_config is shown as below:

Table 3-803. Function timer_break_config

Function name	timer_break_config
Function prototype	void timer_break_config(uint32_t timer_periph, timer_break_parameter_struct* breakpara);
Function descriptions	configure TIMER break function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
breakpara	TIMER break parameter struct, the structure members can refer to Table 3-774, Structure timer break parameter struct.

	<u>3-774. Structure timer break parameter struct</u>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 break function */

timer_break_parameter_struct timer_breakpara;

timer_breakpara.runoffstate      = TIMER_ROS_STATE_DISABLE;
timer_breakpara.ideloffstate     = TIMER_IOS_STATE_DISABLE ;
timer_breakpara.deadtime         = 255;
timer_breakpara.breakpolarity    = TIMER_BREAK_POLARITY_LOW;
timer_breakpara.outputautostate = TIMER_OUTAUTO_ENABLE;
timer_breakpara.protectmode     = TIMER_CCHP_PROT_0;
timer_breakpara.breakstate      = TIMER_BREAK_ENABLE;

timer_break_config(TIMER0, &timer_breakpara);

```

timer_break_enable

The description of timer_break_enable is shown as below:

Table 3-804. Function timer_break_enable

Function name	timer_break_enable
Function prototype	void timer_break_enable(uint32_t timer_periph);
Function descriptions	enable TIMER break function
Precondition	This function can be called only when PROT [1:0] bit-field in TIMERx_CCHP register is 00.
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0,7)	TIMER peripheral selection
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable TIMER0 break function*/
timer_break_enable (TIMER0);
```

timer_break_disable

The description of timer_break_disable is shown as below:

Table 3-805. Function timer_break_disable

Function name	timer_break_disable
Function prototype	void timer_break_disable(uint32_t timer_periph);
Function descriptions	disable TIMER break function
Precondition	This function can be called only when PROT [1:0] bit-field in TIMERx_CCHP register is 00.
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 break function*/
timer_break_disable (TIMER0);
```

timer_automatic_output_enable

The description of timer_automatic_output_enable is shown as below:

Table 3-806. Function timer_automatic_output_enable

Function name	timer_automatic_output_enable
Function prototype	void timer_automatic_output_enable(uint32_t timer_periph);
Function descriptions	enable TIMER output automatic function
Precondition	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable TIMER0 output automatic function */

timer_automatic_output_enable (TIMER0);

```

timer_automatic_output_disable

The description of timer_automatic_output_disable is shown as below:

Table 3-807. Function timer_automatic_output_disable

Function name	timer_automatic_output_disable
Function prototype	void timer_automatic_output_disable (uint32_t timer_periph);
Function descriptions	disable TIMER output automatic function
Precondition	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 output automatic function */

timer_automatic_output_disable (TIMER0);
```

timer_primary_output_config

The description of timer_primary_output_config is shown as below:

Table 3-808. Function timer_primary_output_config

Function name	timer_primary_output_config
Function prototype	void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue);
Function descriptions	configure TIMER primary output function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
newvalue	control value
ENABLE	enable function
DISABLE	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER0 primary output function */

timer_primary_output_config (TIMER0, ENABLE);
```

timer_channel_control_shadow_config

The description of timer_channel_control_shadow_config is shown as below:

Table 3-809. Function timer_channel_control_shadow_config

Function name	timer_channel_control_shadow_config
Function prototype	void timer_channel_control_shadow_config(uint32_t timer_periph, ControlStatus newvalue);
Function descriptions	channel commutation control shadow register enable
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
newvalue	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* channel capture/compare control shadow register enable */

timer_channel_control_shadow_config (TIMER0, ENABLE);
```

timer_channel_control_shadow_update_config

The description of timer_channel_control_shadow_update_config is shown as below:

Table 3-810. Function timer_channel_control_shadow_update_config

Function name	timer_channel_control_shadow_update_config
Function prototype	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint8_t ccuctl);
Function descriptions	configure commutation control shadow register update control
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
ccuctl	channel control shadow register update control
<i>TIMER_UPDATECTL_CCU</i>	the shadow registers update by when CMTG bit is set
<i>TIMER_UPDATECTL_CUTRI</i>	the shadow registers update by when CMTG bit is set or an rising edge of TRGI occurs
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 channel control shadow register update when CMTG bit is set */

timer_channel_control_shadow_update_config (TIMER0, TIMER_UPDATECTL_CCU);

```

timer_channel_output_struct_para_init

The description of timer_channel_output_struct_para_init is shown as below:

Table 3-811. Function timer_channel_output_struct_para_init

Function name	timer_channel_output_struct_para_init
Function prototype	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpara);
Function descriptions	initialize the parameters of TIMER channel output parameter struct with the

	default values
Precondition	-
The called functions	-
Input parameter{in}	
ocpara	TIMER channel output parameter struct, the structure members can refer to Table 3-775. Structure timer_oc_parameter_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER channel output parameter struct with a default value */  
  
timer_oc_parameter_struct timer_ocinitpara;  
  
timer channel output struct para init(timer_ocinitpara);
```

timer_channel_output_config

The description of timer_channel_output_config is shown as below:

Table 3-812. Function timer_channel_output_config

Function name	timer_channel_output_config
Function prototype	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);
Function descriptions	configure TIMER channel output function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMERx(x=0..4,7..13))

<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
<i>ocpara</i>	TIMER channel output parameter struct, the structure members can refer to Table 3-775. Structure timer_oc_parameter_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output function */

timer_oc_parameter_struct timer_ocinitpara;

timer_ocinitpara.outputstate = TIMER_CCX_ENABLE;

timer_ocinitpara.outputnstate = TIMER_CCXN_ENABLE;

timer_ocinitpara.ocpolarity = TIMER_OC_POLARITY_HIGH;

timer_ocinitpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;

timer_ocinitpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;

timer_ocinitpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;

timer_channel_output_config(TIMER0, TIMER_CH_0, &timer_ocinitpara);
```

timer_channel_output_mode_config

The description of `timer_channel_output_mode_config` is shown as below:

Table 3-813. Function `timer_channel_output_mode_config`

Function name	<code>timer_channel_output_mode_config</code>
Function prototype	<code>void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);</code>
Function descriptions	configure TIMER channel output compare mode
Precondition	-
The called functions	-

Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (<i>TIMERx</i> (x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (<i>TIMERx</i> (x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (<i>TIMERx</i> (x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (<i>TIMERx</i> (x=0..4,7))
Input parameter{in}	
ocmode	channel output compare mode
<i>TIMER_OC_MODE_TIMING</i>	timing mode
<i>TIMER_OC_MODE_ACTIVE</i>	set the channel output
<i>TIMER_OC_MODE_INACTIVE</i>	clear the channel output
<i>TIMER_OC_MODE_TOGGLE</i>	toggle on match
<i>TIMER_OC_MODE_LOW</i>	force low mode
<i>TIMER_OC_MODE_HIGH</i>	force high mode
<i>TIMER_OC_MODE_PWM0</i>	PWM mode 0
<i>TIMER_OC_MODE_PWM1</i>	PWM mode 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 channel PWM 0 mode */

timer_channel_output_mode_config(TIMER0, TIMER_CH_0, TIMER_OC_MODE_PWM0);

```

timer_channel_output_pulse_value_config

The description of timer_channel_output_pulse_value_config is shown as below:

Table 3-814. Function timer_channel_output_pulse_value_config

Function name	timer_channel_output_pulse_value_config
Function prototype	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);
Function descriptions	configure TIMER channel output pulse value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (<i>TIMERx</i> (x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (<i>TIMERx</i> (x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (<i>TIMERx</i> (x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (<i>TIMERx</i> (x=0..4,7))
Input parameter{in}	
pulse	channel output pulse value (0~65535)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 channel 0 output pulse value */

```

```
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

timer_channel_output_shadow_config

The description of timer_channel_output_shadow_config is shown as below:

Table 3-815. Function timer_channel_output_shadow_config

Function name	timer_channel_output_shadow_config
Function prototype	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
Function descriptions	configure TIMER channel output shadow function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
ocshadow	channel output shadow state
<i>TIMER_OC_SHADOW_ENABLE</i>	channel output shadow state enable
<i>TIMER_OC_SHADOW_DISABLE</i>	channel output shadow state disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/*configure TIMER0 channel 0 output shadow function */

timer_channel_output_shadow_config (TIMER0, TIMER_CH_0,
TIMER_OC_SHADOW_ENABLE);

```

timer_channel_output_fast_config

The description of timer_channel_output_fast_config is shown as below:

Table 3-816. Function timer_channel_output_fast_config

Function name	timer_channel_output_fast_config
Function prototype	void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
Function descriptions	configure TIMER channel output fast function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (<i>TIMERx(x=0..4,7..13)</i>)
<i>TIMER_CH_1</i>	TIMER channel 1 (<i>TIMERx(x=0..4,7,8,11)</i>)
<i>TIMER_CH_2</i>	TIMER channel 2 (<i>TIMERx(x=0..4,7)</i>)
<i>TIMER_CH_3</i>	TIMER channel 3 (<i>TIMERx(x=0..4,7)</i>)
Input parameter{in}	
ocfast	channel output fast function
<i>TIMER_OC_FAST_ENA_BLE</i>	channel output fast function enable
<i>TIMER_OC_FAST_DISABLE</i>	channel output fast function disable
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output fast function */

timer_channel_output_fast_config (TIMER0, TIMER_CH_0, TIMER_OC_FAST_ENABLE);
```

timer_channel_output_clear_config

The description of timer_channel_output_clear_config is shown as below:

Table 3-817. Function timer_channel_output_clear_config

Function name	timer_channel_output_clear_config
Function prototype	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
Function descriptions	configure TIMER channel output clear function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER periphera
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (<i>TIMERx</i> (x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (<i>TIMERx</i> (x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (<i>TIMERx</i> (x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (<i>TIMERx</i> (x=0..4,7))
Input parameter{in}	
occlear	channel output clear function
<i>TIMER_OC_CLEAR_ENABLE</i>	channel output clear function enable
<i>TIMER_OC_CLEAR_DISABLE</i>	channel output clear function disable

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output clear function */
timer_channel_output_clear_config (TIMER0, TIMER_CH_0,
TIMER_OC_CLEAR_ENABLE);
```

timer_channel_output_polarity_config

The description of timer_channel_output_polarity_config is shown as below:

Table 3-818. Function timer_channel_output_polarity_config

Function name	timer_channel_output_polarity_config
Function prototype	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
Function descriptions	configure TIMER channel output polarity
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMERx(x=0..4,7..13))
TIMER_CH_1	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
TIMER_CH_2	TIMER channel 2 (TIMERx(x=0..4,7))
TIMER_CH_3	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
ocpolarity	channel output polarity
TIMER_OC_POLARITY	channel output polarity is high

<i>_HIGH</i>	
<i>TIMER_OC_POLARITY_LOW</i>	channel output polarity is low
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output polarity */

timer_channel_output_polarity_config (TIMER0, TIMER_CH_0,
TIMER_OC_POLARITY_HIGH);
```

timer_channel_complementary_output_polarity_config

The description of timer_channel_complementary_output_polarity_config is shown as below:

Table 3-819. Function timer_channel_complementary_output_polarity_config

Function name	timer_channel_complementary_output_polarity_config
Function prototype	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnpolarity);
Function descriptions	configure TIMER channel complementary output polarity
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<i>TIMER_CH_2</i>	TIMER channel 2
Input parameter{in}	

ocpolarity	channel complementary output polarity
<i>TIMER_OCN_POLARITY_HIGH</i>	channel complementary output polarity is high
<i>TIMER_OCN_POLARITY_LOW</i>	channel complementary output polarity is low
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 channel 0 complementary output polarity */
timer_channel_complementary_output_polarity_config (TIMER0, TIMER_CH_0,
TICKER_OCN_POLARITY_HIGH);

```

timer_channel_output_state_config

The description of timer_channel_output_state_config is shown as below:

Table 3-820. Function timer_channel_output_state_config

Function name	timer_channel_output_state_config
Function prototype	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
Function descriptions	configure TIMER channel enable state
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (<i>TIMERx(x=0..4,7..13)</i>)
<i>TIMER_CH_1</i>	TIMER channel 1 (<i>TIMERx(x=0..4,7,8,11)</i>)

<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
state	TIMER channel enable state
<i>TIMER_CCX_ENABLE</i>	channel enable
<i>TIMER_CCX_DISABLE</i>	channel disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 enable state */
timer_channel_output_state_config (TIMER0, TIMER_CH_0, TIMER_CCX_ENABLE);
```

timer_channel_complementary_output_state_config

The description of timer_channel_complementary_output_state_config is shown as below:

Table 3-821. Function timer_channel_complementary_output_state_config

Function name	timer_channel_complementary_output_state_config
Function prototype	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
Function descriptions	configure TIMER channel complementary output enable state
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0

<i>TIMER_CH_1</i>	TIMER channel 1
<i>TIMER_CH_2</i>	TIMER channel 2
Input parameter{in}	
state	TIMER channel complementary output enable state
<i>TIMER_CCXN_ENABLE</i>	channel complementary enable
<i>TIMER_CCXN_DISABLE</i>	channel complementary disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 channel 0 complementary output enable state */

timer_channel_complementary_output_state_config (TIMER0, TIMER_CH_0,
TICKER_CCXN_ENABLE);

```

timer_channel_input_struct_para_init

The description of timer_channel_input_struct_para_init is shown as below:

Table 3-822. Function timer_channel_input_struct_para_init

Function name	timer_channel_input_struct_para_init
Function prototype	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
Function descriptions	initialize the parameters of TIMER channel input parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
icpara	TIMER channel input parameter struct, the structure members can refer to Table 3-776. Structure timer_ic_parameter_struct .
Output parameter{out}	
-	-

Return value	
-	-

Example:

```

/* initialize TIMER channel input parameter struct with a default value */

timer_ic_parameter_struct timer_icinitpara;

timer_channel_input_struct_para_init(timer_icinitpara);

```

timer_input_capture_config

The description of timer_input_capture_config is shown as below:

Table 3-823. Function timer_input_capture_config

Function name	timer_input_capture_config
Function prototype	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
Function descriptions	configure TIMER input capture parameter
Precondition	-
The called functions	timer_channel_input_capture_prescaler_config
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMERx(x=0..4,7..13))
TIMER_CH_1	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
TIMER_CH_2	TIMER channel 2 (TIMERx(x=0..4,7))
TIMER_CH_3	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
icpara	TIMER channel input parameter struct, the structure members can refer to Table 3-776. Structure timer_ic_parameter_struct .
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure TIMER0 input capture parameter */
timer_ic_parameter_struct timer_icinitpara;
timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;
timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
```

timer_channel_input_capture_prescaler_config

The description of timer_channel_input_capture_prescaler_config is shown as below:

Table 3-824. Function timer_channel_input_capture_prescaler_config

Function name	timer_channel_input_capture_prescaler_config
Function prototype	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);
Function descriptions	configure TIMER channel input capture prescaler value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))

Input parameter{in}	
prescaler	channel input capture prescaler value
<i>TIMER_IC_PSC_DIV1</i>	no prescaler
<i>TIMER_IC_PSC_DIV2</i>	divided by 2
<i>TIMER_IC_PSC_DIV4</i>	divided by 4
<i>TIMER_IC_PSC_DIV8</i>	divided by 8
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 input capture prescaler value */
timer_channel_input_capture_prescaler_config (TIMER0, TIMER_CH_0,
TICKER_IC_PSC_DIV2);
```

timer_channel_capture_value_register_read

The description of timer_channel_capture_value_register_read is shown as below:

Table 3-825. Function timer_channel_capture_value_register_read

Function name	timer_channel_capture_value_register_read
Function prototype	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
Function descriptions	read TIMER channel capture compare register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))

<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Output parameter{out}	
-	-
Return value	
uint32_t	channel capture compare register value (0x0000~0xFFFF)

Example:

```

/* read TIMER0 channel 0 capture compare register value */

uint32_t ch0_value = 0;

ch0_value = timer_channel_capture_value_register_read (TIMER0, TIMER_CH_0);

```

timer_input_pwm_capture_config

The description of `timer_input_pwm_capture_config` is shown as below:

Table 3-826. Function `timer_input_pwm_capture_config`

Function name	timer_input_pwm_capture_config
Function prototype	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
Function descriptions	configure TIMER input pwm capture function
Precondition	-
The called functions	timer_channel_input_capture_prescaler_config
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
Input parameter{in}	

icpwm	TIMER channel input pwm parameter struct, the structure members can refer to Table 3-776. Structure timer_ic_parameter_struct.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 input pwm capture parameter */
timer_ic_parameter_struct timer_icinitpara;
timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;
timer_input_pwm_capture_config (TIMER0, TIMER_CH_0, &timer_icinitpara);

```

timer_hall_mode_config

The description of timer_hall_mode_config is shown as below:

Table 3-827. Function timer_hall_mode_config

Function name	timer_hall_mode_config
Function prototype	void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode);
Function descriptions	configure TIMER hall sensor mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
Input parameter{in}	
hallmode	TIMER hall sensor mode state
<i>TIMER_HALLINTERFA CE_ENABLE</i>	TIMER hall sensor mode enable

<i>TIMER_HALLINTERFA CE_DISABLE</i>	TIMER hall sensor mode disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 hall sensor mode */
timer_hall_mode_config (TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

timer_input_trigger_source_select

The description of timer_input_trigger_source_select is shown as below:

Table 3-828. Function timer_input_trigger_source_select

Function name	timer_input_trigger_source_select
Function prototype	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);
Function descriptions	select TIMER input trigger source
Precondition	SMC[2:0] = 000
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
intrigger	trigger selection
<i>TIMER_SMCFG_TRGS EL_ITI0</i>	Internal trigger input 0 (ITI0, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS EL_ITI1</i>	Internal trigger input 0 (ITI1, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS EL_ITI2</i>	Internal trigger input 0 (ITI2, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS EL_ITI3</i>	Internal trigger input 0 (ITI3, TIMERx(x=0..4,7,8,11))

<i>EL_ITI3</i>	
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI0F_ED</i>	CI0 edge flag (CI0F_ED, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI0FE0</i>	channel 0 input Filtered output (CI0FE0, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI1FE1</i>	channel 1 input Filtered output (CI1FE1, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS</i> <i>EL_ETIFP</i>	External trigger input filter output(ETIFP, TIMERx(x=0..4,7))
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* select TIMER0 input trigger source */

timer_input_trigger_source_select (TIMER0, TIMER_SMCFG_TRGSEL_ITIO);

```

timer_master_output_trigger_source_select

The description of timer_master_output_trigger_source_select is shown as below:

Table 3-829. Function timer_master_output_trigger_source_select

Function name	timer_master_output_trigger_source_select
Function prototype	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
Function descriptions	select TIMER master mode output trigger source
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
outrigger	master mode control

<i>TIMER_TRI_OUT_SRCRESET</i>	Reset. When the UPG bit in the TIMERx_SWEVG register is set or a reset is generated by the slave mode controller, a TRGO pulse occurs. And in the latter case, the signal on TRGO is delayed compared to the actual reset
<i>TIMER_TRI_OUT_SRC_ENABLE</i>	Enable. This mode is useful to start several timers at the same time or to control a window in which a slave timer is enabled. In this mode the master mode controller selects the counter enable signal as TRGO. The counter enable signal is set when CEN control bit is set or the trigger input in pause mode is high. There is a delay between the trigger input in pause mode and the TRGO output, except if the master-slave mode is selected.
<i>TIMER_TRI_OUT_SRC_UPDATE</i>	Update. In this mode the master mode controller selects the update event as TRGO.
<i>TIMER_TRI_OUT_SRC_CC0</i>	Capture/compare pulse. In this mode the master mode controller generates a TRGO pulse when a capture or a compare match occurred in channel 0.
<i>TIMER_TRI_OUT_SRC_O0CPRE</i>	Compare. In this mode the master mode controller selects the O0CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O1CPRE</i>	Compare. In this mode the master mode controller selects the O1CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O2CPRE</i>	Compare. In this mode the master mode controller selects the O2CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O3CPRE</i>	Compare. In this mode the master mode controller selects the O3CPRE signal is used as TRGO.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select TIMER0 master mode output trigger source */
timer_master_output_trigger_source_select (TIMER0, TIMER_TRI_OUT_SRC_RESET);
```

timer_slave_mode_select

The description of timer_slave_mode_select is shown as below:

Table 3-830. Function timer_slave_mode_select

Function name	timer_slave_mode_select

Function prototype	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
Function descriptions	select TIMER slave mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
Input parameter{in}	
slavemode	slave mode
<i>TIMER_SLAVE_MODE_DISABLE</i>	slave mode disable
<i>TIMER_QUAD_DECODER_MODE0</i>	quadrature decoder mode 0
<i>TIMER_QUAD_DECODER_MODE1</i>	quadrature decoder mode 1
<i>TIMER_QUAD_DECODER_MODE2</i>	quadrature decoder mode 2
<i>TIMER_SLAVE_MODE_RESTART</i>	restart mode
<i>TIMER_SLAVE_MODE_PAUSE</i>	pause mode
<i>TIMER_SLAVE_MODE_EVENT</i>	event mode
<i>TIMER_SLAVE_MODE_EXTERNAL0</i>	external clock mode 0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select TIMER0 slave mode */
```

```
timer_slave_mode_select (TIMER0, TIMER_QUAD_DECODER_MODE0);
```

timer_master_slave_mode_config

The description of timer_master_slave_mode_config is shown as below:

Table 3-831. Function timer_master_slave_mode_config

Function name	timer_master_slave_mode_config
Function prototype	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);
Function descriptions	configure TIMER master slave mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..4,7,8,11)	TIMER peripheral selection
Input parameter{in}	
masterslave	master slave mode state
TIMER_MASTER_SLAVE_MODE_ENABLE	master slave mode enable
TIMER_MASTER_SLAVE_MODE_DISABLE	master slave mode disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 master slave mode */  
  
timer_master_slave_mode_config (TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

timer_external_trigger_config

The description of timer_external_trigger_config is shown as below:

Table 3-832. Function timer_external_trigger_config

Function name	timer_external_trigger_config
Function prototype	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
Function descriptions	configure TIMER external trigger input
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
Input parameter{in}	
extprescaler	external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
Input parameter{in}	
expolarity	external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
Input parameter{in}	
extfilter	external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 external trigger input */

timer_external_trigger_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
    TIMER_ETP_FALLING, 10);
```

timer_quadrature_decoder_mode_config

The description of timer_quadrature_decoder_mode_config is shown as below:

Table 3-833. Function timer_quadrature_decoder_mode_config

Function name	timer_quadrature_decoder_mode_config
Function prototype	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
Function descriptions	configure TIMER quadrature decoder mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
Input parameter{in}	
decomode	quadrature decoder mode
<i>TIMER_QUAD_DECODER_MODE0</i>	counter counts on CI0FE0 edge depending on CI1FE1 level
<i>TIMER_QUAD_DECODER_MODE1</i>	counter counts on CI1FE1 edge depending on CI0FE0 level
<i>TIMER_QUAD_DECODER_MODE2</i>	counter counts on both CI0FE0 and CI1FE1 edges depending on the level of the other input
Input parameter{in}	
ic0polarity	IC0 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
Input parameter{in}	

ic1polarity	IC1 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 quadrature decoder mode */
timer_quadrature_decoder_mode_config (TIMER0, TIMER_QUAD_DECODER_MODE0,
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

timer_internal_clock_config

The description of timer_internal_clock_config is shown as below:

Table 3-834. Function timer_internal_clock_config

Function name	timer_internal_clock_config
Function prototype	void timer_internal_clock_config(uint32_t timer_periph);
Function descriptions	configure TIMER internal clock mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 internal clock mode */
```

```
timer_internal_clock_config (TIMER0);
```

timer_internal_trigger_as_external_clock_config

The description of `timer_internal_trigger_as_external_clock_config` is shown as below:

Table 3-835. Function `timer_internal_trigger_as_external_clock_config`

Function name	timer_internal_trigger_as_external_clock_config
Function prototype	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
Function descriptions	configure TIMER the internal trigger as external clock input
Precondition	-
The called functions	timer_input_trigger_source_select
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
Input parameter{in}	
intrigger	trigger selection
<i>TIMER_SMCFG_TRGS_EL_ITI0</i>	Internal trigger input 0 (ITI0)
<i>TIMER_SMCFG_TRGS_EL_ITI1</i>	Internal trigger input 0 (ITI1)
<i>TIMER_SMCFG_TRGS_EL_ITI2</i>	Internal trigger input 0 (ITI2)
<i>TIMER_SMCFG_TRGS_EL_ITI3</i>	Internal trigger input 0 (ITI3)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 the internal trigger ITI0 as external clock input */
```

timer_internal_trigger_as_external_clock_config (TIMERO, TIMER_SMCFG_TRGSEL_ITI0);

timer_external_trigger_as_external_clock_config

The description of timer_external_trigger_as_external_clock_config is shown as below:

Table 3-836. Function timer_external_trigger_as_external_clock_config

Function name	timer_external_trigger_as_external_clock_config
Function prototype	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t exttrigger, uint16_t expolarity, uint32_t extfilter);
Function descriptions	configure TIMER the external trigger as external clock input
Precondition	-
The called functions	timer_input_trigger_source_select
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..4,7,8,11)	TIMER peripheral selection
Input parameter{in}	
exttrigger	external trigger selection
TIMER_SMCFG_TRGS_EL_CI0F_ED	CI0 edge flag (CI0F_ED)
TIMER_SMCFG_TRGS_EL_CI0FE0	channel 0 input Filtered output (CI0FE0)
TIMER_SMCFG_TRGS_EL_CI1FE1	channel 1 input Filtered output (CI1FE1)
Input parameter{in}	
expolarity	external trigger polarity
TIMER_IC_POLARITY_RISING	active high or rising edge active
TIMER_IC_POLARITY_FALLING	active low or falling edge active
Input parameter{in}	
extfilter	external trigger filter control (0~15)
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* configure TIMER0 the external trigger CI0FE0 as external clock input */

timer_external_trigger_as_external_clock_config (TIMER0,
TIMER_SMCFG_TRGSEL_CI0FE0, TIMER_IC_POLARITY_RISING, 0);
```

timer_external_clock_mode0_config

The description of timer_external_clock_mode0_config is shown as below:

Table 3-837. Function timer_external_clock_mode0_config

Function name	timer_external_clock_mode0_config
Function prototype	void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
Function descriptions	configure TIMER the external clock mode0
Precondition	-
The called functions	timer_external_trigger_config
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
Input parameter{in}	
extprescaler	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
Input parameter{in}	

expolarity	ETI external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
Input parameter{in}	
extfilter	ETI external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 the external clock mode0 */

timer_external_clock_mode0_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 0);
```

timer_external_clock_mode1_config

The description of timer_external_clock_mode1_config is shown as below:

Table 3-838. Function timer_external_clock_mode1_config

Function name	timer_external_clock_mode1_config
Function prototype	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
Function descriptions	configure TIMER the external clock mode1
Precondition	-
The called functions	timer_external_trigger_config
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
Input parameter{in}	
extprescaler	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided

<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
Input parameter{in}	
expolarity	ETI external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
Input parameter{in}	
extfilter	ETI external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 the external clock mode1 */
timer_external_clock_mode1_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TICKER_ETP_FALLING, 0);
```

timer_external_clock_mode1_disable

The description of timer_external_clock_mode1_disable is shown as below:

Table 3-839. Function timer_external_clock_mode1_disable

Function name	timer_external_clock_mode1_disable
Function prototype	void timer_external_clock_mode1_disable(uint32_t timer_periph);
Function descriptions	disable TIMER the external clock mode1
Precondition	-
The called functions	-
Input parameter{in}	

timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 the external clock mode1 */
timer_external_clock_mode1_disable (TIMER0);
```

timer_channel_remap_config

The description of timer_channel_remap_config is shown as below:

Table 3-840. Function timer_channel_remap_config

Function name	timer_channel_remap_config
Function prototype	void timer_channel_remap_config(uint32_t timer_periph, uint32_t remap);
Function descriptions	configure TIMER channel remap function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=1,4,10)</i>	TIMER peripheral selection
Input parameter{in}	
remap	TIMER remap selection
<i>TIMER1_ITI1_RMP_TIM_ER7_TRGO</i>	timer1 internal trigger input1 remap to TIMER7_TRGO
<i>TIMER1_ITI1_RMP_ET_HERNET_PTP</i>	timer1 internal trigger input1 remap to ethernet PTP
<i>TIMER1_ITI1_RMP_US_B_FS_SOF</i>	timer1 internal trigger input1 remap to USB FS SOF
<i>TIMER1_ITI1_RMP_US</i>	timer1 internal trigger input1 remap to USB HS SOF

<i>B_HS_SOF</i>	
<i>TIMER4_CI3_RMP_GPIO</i>	timer4 channel 3 input remap to GPIO pin
<i>TIMER4_CI3_RMP_IRC32K</i>	timer4 channel 3 input remap to IRC32K
<i>TIMER4_CI3_RMP_LXTAL</i>	timer4 channel 3 input remap to LXTAL
<i>TIMER4_CI3_RMP_RTC_WAKEUP_INT</i>	timer4 channel 3 input remap to RTC wakeup interrupt
<i>TIMER10_ITI1_RMP_GPIO</i>	internal trigger input1 remap based on GPIO setting
<i>TIMER10_ITI1_RMP_RTC_HXTAL_DIV</i>	timer10 internal trigger input1 remap HXTAL _DIV(clock used for RTC which is HXTAL clock divided by RTCDIV bits in RCU_CFG0 register)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure timer1 internal trigger input1 remap to TIMER7_TRGO */
timer_channel_remap_config (TIMER1,TIMER1_ITI1_RMP_TIMER7_TRGO);
```

timer_write_chxval_register_config

The description of **timer_write_chxval_register_config** is shown as below:

Table 3-841. Function timer_write_chxval_register_config

Function name	timer_write_chxval_register_config
Function prototype	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
Function descriptions	configure TIMER write CHxVAL register selection
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral

<i>TIMERx(x=0..4,7..13)</i>	TIMER peripheral selection
Input parameter{in}	
ccsel	write CHxVAL register selection
<i>TIMER_CHVSEL_DISA BLE</i>	no effect
<i>TIMER_CHVSEL_ENAB LE</i>	when write the CHxVAL register, if the write value is same as the CHxVAL value, the write access is ignored
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 write CHxVAL register selection */
timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

timer_output_value_selection_config

The description of `timer_output_value_selection_config` is shown as below:

Table 3-842. Function `timer_output_value_selection_config`

Function name	timer_output_value_selection_config
Function prototype	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);
Function descriptions	configure TIMER output value selection
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx (x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
outsel	output value selection
<i>TIMER_OUTSEL_DISA</i>	no effect

<i>BLE</i>	
<i>TIMER_OUTSEL_ENAB</i> <i>LE</i>	if POEN and IOS is 0, the output disabled
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER output value selection */

timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

timer_flag_get

The description of timer_flag_get is shown as below:

Table 3-843. Function timer_flag_get

Function name	timer_flag_get
Function prototype	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
Function descriptions	get TIMER flags
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
flag	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag,TIMERx(x=0..13)
<i>TIMER_FLAG_CH0</i>	channel 0 flag,TIMERx(x=0..4,7..13)
<i>TIMER_FLAG_CH1</i>	channel 1 flag,TIMERx(x=0..4,7,8,11)
<i>TIMER_FLAG_CH2</i>	channel 2 flag,TIMERx(x=0..4,7)
<i>TIMER_FLAG_CH3</i>	channel 3 flag,TIMERx(x=0..4,7)
<i>TIMER_FLAG_CMT</i>	channel commutation flag,TIMERx(x=0,7)

<i>TIMER_FLAG_TRG</i>	trigger flag,TIMERx(x=0,7,8,11)
<i>TIMER_FLAG_BRK</i>	break flag,TIMERx(x=0,7)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag,TIMERx(x=0..4,7..11)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag,TIMERx(x=0..4,7,8,11)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag,TIMERx(x=0..4,7)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag,TIMERx(x=0..4,7)
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get TIMER0 update flags */
FlagStatus Flag_status = RESET;
Flag_status = timer_flag_get (TIMER0, TIMER_FLAG_UP);
```

timer_flag_clear

The description of **timer_flag_clear** is shown as below:

Table 3-844. Function timer_flag_clear

Function name	timer_flag_clear
Function prototype	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
Function descriptions	clear TIMER flags
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
flag	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag,TIMERx(x=0..13)

<i>TIMER_FLAG_CH0</i>	channel 0 flag,TIMERx(x=0..4,7..13)
<i>TIMER_FLAG_CH1</i>	channel 1 flag,TIMERx(x=0..4,7,8,11)
<i>TIMER_FLAG_CH2</i>	channel 2 flag,TIMERx(x=0..4,7)
<i>TIMER_FLAG_CH3</i>	channel 3 flag,TIMERx(x=0..4,7)
<i>TIMER_FLAG_CMT</i>	channel commutation flag,TIMERx(x=0,7)
<i>TIMER_FLAG_TRG</i>	trigger flag,TIMERx(x=0,7,8,11)
<i>TIMER_FLAG_BRK</i>	break flag,TIMERx(x=0,7)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag,TIMERx(x=0..4,7..11)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag,TIMERx(x=0..4,7,8,11)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag,TIMERx(x=0..4,7)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag,TIMERx(x=0..4,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TIMER0 update flags */

timer_flag_clear (TIMER0, TIMER_FLAG_UP);
```

timer_interrupt_enable

The description of timer_interrupt_enable is shown as below:

Table 3-845. Function timer_interrupt_enable

Function name	timer_interrupt_enable
Function prototype	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
Function descriptions	enable the TIMER interrupt
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral

<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<i>interrupt</i>	timer interrupt enable source
<i>TIMER_INT_UP</i>	update interrupt enable, TIMERx (x=0..13)
<i>TIMER_INT_CH0</i>	channel 0 interrupt enable, TIMERx (x=0..4,7..13)
<i>TIMER_INT_CH1</i>	channel 1 interrupt enable, TIMERx (x=0..4,7,8,11)
<i>TIMER_INT_CH2</i>	channel 2 interrupt enable, TIMERx (x=0..4,7)
<i>TIMER_INT_CH3</i>	channel 3 interrupt enable , TIMERx (x=0..4,7)
<i>TIMER_INT_CMT</i>	commutation interrupt enable, TIMERx (x=0,7)
<i>TIMER_INT_TRG</i>	trigger interrupt enable, TIMERx (x=0..4,7,8,11)
<i>TIMER_INT_BRK</i>	break interrupt enable, TIMERx (x=0,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TIMER0 update interrupt */

timer_interrupt_enable (TIMER0, TIMER_INT_UP);
```

timer_interrupt_disable

The description of timer_interrupt_disable is shown as below:

Table 3-846. Function timer_interrupt_disable

Function name	timer_interrupt_disable
Function prototype	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
Function descriptions	disable the TIMER interrupt
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral

<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
interrupt	timer interrupt disable source
<i>TIMER_INT_UP</i>	update interrupt disable, TIMERx(x=0..13)
<i>TIMER_INT_CH0</i>	channel 0 interrupt disable, TIMERx(x=0..4,7..13)
<i>TIMER_INT_CH1</i>	channel 1 interrupt disable, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_CH2</i>	channel 2 interrupt disable, TIMERx(x=0..4,7)
<i>TIMER_INT_CH3</i>	channel 3 interrupt disable, TIMERx(x=0..4,7)
<i>TIMER_INT_CMT</i>	commutation interrupt disable, TIMERx(x=0,7)
<i>TIMER_INT_TRG</i>	trigger interrupt disable, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_BRK</i>	break interrupt disable, TIMERx(x=0,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 update interrupt */

timer_interrupt_disable (TIMER0, TIMER_INT_UP);
```

timer_interrupt_flag_get

The description of timer_interrupt_flag_get is shown as below:

Table 3-847. Function timer_interrupt_flag_get

Function name	timer_interrupt_flag_get
Function prototype	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t interrupt);
Function descriptions	get timer interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	

timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
interrupt	the timer interrupt bits
TIMER_INT_FLAG_UP	update interrupt flag,TIMERx(x=0..13)
TIMER_INT_FLAG_CH0	channel 0 interrupt flag,TIMERx(x=0..4,7..13)
TIMER_INT_FLAG_CH1	channel 1 interrupt flag,TIMERx(x=0..4,7,8,11)
TIMER_INT_FLAG_CH2	channel 2 interrupt flag,TIMERx(x=0..4,7)
TIMER_INT_FLAG_CH3	channel 3 interrupt flag,TIMERx(x=0..4,7)
TIMER_INT_FLAG_CM_T	channel commutation interrupt flag,TIMERx(x=0,7)
TIMER_INT_FLAG_TRG	trigger interrupt flag,TIMERx(x=0,7,8,11)
TIMER_INT_FLAG_BRK	break interrupt flag,TIMERx(x=0,7)
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get TIMER0 update interrupt flag */

FlagStatus Flag_interrupt = RESET;

Flag_interrupt = timer_interrupt_flag_get (TIMER0, TIMER_INT_FLAG_UP);
```

timer_interrupt_flag_clear

The description of `timer_interrupt_flag_clear` is shown as below:

Table 3-848. Function `timer_interrupt_flag_clear`

Function name	<code>timer_interrupt_flag_clear</code>
Function prototype	<code>void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt);</code>
Function descriptions	clear TIMER interrupt flag
Precondition	-

The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
interrupt	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag,TIMERx(x=0..13)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag,TIMERx(x=0..4,7..13)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag,TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag,TIMERx(x=0..4,7)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag,TIMERx(x=0..4,7)
<i>TIMER_INT_FLAG_CM_T</i>	channel commutation interrupt flag,TIMERx(x=0,7)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag,TIMERx(x=0,7,8,11)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag,TIMERx(x=0,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TIMER0 update interrupt flag */
timer_interrupt_flag_clear (TIMER0, TIMER_INT_FLAG_UP);
```

3.27. TLI

The TLI (TFT-LCD Interface) module handles the synchronous LCD interface and provides pixel data, clock and timing signals for passive LCD display. The TLI registers are listed in chapter [3.27.1](#), the TLI firmware functions are introduced in chapter [3.27.2](#).

3.27.1. Descriptions of Peripheral registers

TLI registers are listed in the table shown as below:

Table 3-849. TLI Registers

Registers	Descriptions
TLI_SPSZ	TLI synchronous pulse size register
TLI_BPSZ	TLI back-porch size register
TLI_ASZ	TLI active size register
TLI_TSZ	TLI total size register
TLI_CTL	TLI control register
TLI_RL	TLI reload Layer register
TLI_BGC	TLI background color register
TLI_INTEN	TLI interrupt enable register
TLI_INTF	TLI interrupt flag register
TLI_INTC	TLI interrupt flag clear register
TLI_LM	TLI line mark register
TLI_CPOOS	TLI current pixel position register
TLI_STAT	TLI status register
TLI_LxCTL	TLI layer x control register
TLI_LxHPOS	TLI layer x horizontal position parameters register
TLI_LxVPOS	TLI layer x vertical position parameters register
TLI_LxCKEY	TLI layer x color key register
TLI_LxPPF	TLI layer x packeted pixel format register
TLI_LxSA	TLI layer x specified alpha register
TLI_LxDC	TLI layer x default color register
TLI_LxBLEND	TLI layer x blending register
TLI_LxFBADDR	TLI layer x frame base address register
TLI_LxFLLEN	TLI layer x frame line length register
TLI_LxFTLN	TLI layer x frame total line number register
TLI_LxLUT	TLI layer x Look Up Table register

3.27.2. Descriptions of Peripheral functions

TLI firmware functions are listed in the table shown as below:

Table 3-850. TLI firmware function

Function name	Function description
tli_deinit	deinitialize TLI registers
tli_struct_para_init	initialize the parameters of TLI parameter structure with the default values, it is suggested that call this function after a tli_parameter_struct structure is defined
tli_init	initialize TLI
tli_dither_config	configure TLI dither function
tli_enable	enable TLI
tli_disable	disable TLI
tli_reload_config	configurate TLI reload mode
tli_layer_struct_para_init	initialize the parameters of TLI layer structure with the default values, it is suggested that call this function after a tli_layer_parameter_struct structure is defined
tli_layer_init	initialize TLI layer
tli_layer_window_offset_modify	reconfigure window position
tli_lut_struct_para_init	initialize the parameters of TLI layer LUT structure with the default values, it is suggested that call this function after a tli_layer_lut_parameter_struct structure is defined
tli_lut_init	initialize TLI layer LUT
tli_color_key_init	initialize TLI layer color key
tli_layer_enable	enable TLI layer
tli_layer_disable	disable TLI layer
tli_color_key_enable	enable TLI layer color keying
tli_color_key_disable	disable TLI layer color keying
tli_lut_enable	enable TLI layer LUT
tli_lut_disable	disable TLI layer LUT
tli_line_mark_set	set line mark value

Function name	Function description
tli_current_pos_get	get current displayed position
tli_interrupt_enable	enable TLI interrupt
tli_interrupt_disable	disable TLI interrupt
tli_interrupt_flag_get	get TLI interrupt flag
tli_interrupt_flag_clear	clear TLI interrupt flag
tli_flag_get	get TLI flag or state in TLI_INTF register or TLI_STAT register

Structure tli_parameter_struct

Table3-851. Structure tli_parameter_struct

Member name	Function description
synpsz_vpsz	size of the vertical synchronous pulse
synpsz_hpsz	size of the horizontal synchronous pulse
backpsz_vbpsz	size of the vertical back porch plus synchronous pulse
backpsz_hbpsz	size of the horizontal back porch plus synchronous pulse
activesz_vasz	size of the vertical active area width plus back porch and synchronous pulse
activesz_hasz	size of the horizontal active area width plus back porch and synchronous pulse
totalsz_vtsz	vertical total size of the display
totalsz_htsz	horizontal total size of the display
backcolor_red	background value red
backcolor_green	background value green
backcolor_blue	background value blue
signalpolarity_hs	horizontal pulse polarity selection
signalpolarity_vs	vertical pulse polarity selection
signalpolarity_de	data enable polarity selection
signalpolarity_pixelck	pixel clock polarity selection

Structure tli_layer_parameter_struct

Table3-852. Structure tli_layer_parameter_struct

Member name	Function description
layer_window_rightpos	window right position
layer_window_leftpos	window left position
layer_window_bottompos	window bottom position
layer_window_toppos	window top position
layer_ppf	packetized pixel format
layer_sa	specified alpha
layer_default_alpha	the default color alpha
layer_default_red	the default color red
layer_default_green	the default color green
layer_default_blue	the default color blue
layer_acf1	alpha calculation factor 1 of blending method
layer_acf2	alpha calculation factor 2 of blending method
layer_frame_bufaddr	frame buffer base address
layer_frame_buf_stride_offset	frame buffer stride offset
layer_frame_line_length	frame line length
layer_frame_total_line_number	frame total line number

Structure tli_layer_lut_parameter_struct

Table3-853. Structure tli_layer_lut_parameter_struct

Member name	Function description
layer_table_addr	look up table write address
layer_lut_channel_red	red channel of a LUT entry
layer_lut_channel_green	green channel of a LUT entry
layer_lut_channel_blue	blue channel of a LUT entry

tli_deinit

The description of tli_deinit is shown as below:

Table 3-854. Function tli_deinit

Function name	tli_deinit
Function prototype	void tli_deinit(void);
Function descriptions	deinitialize TLI registers
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize TLI */

tli_deinit();
```

tli_struct_para_init

The description of tli_struct_para_init is shown as below:

Table 3-855. Function tli_struct_para_init

Function name	tli_struct_para_init
Function prototype	void tli_struct_para_init(tli_parameter_struct *tli_struct);
Function descriptions	initialize the parameters of TLI parameter structure with the default values, it is suggested that call this function after a tli_parameter_struct structure is defined
Precondition	-
The called functions	-
Input parameter{in}	
*tli_struct	a pointer to tli_parameter_struct
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
tli_parameter_struct tli_init_struct;
/* initialize the parameters of TLI parameter structure with the default values */
tli_struct_para_init(&tli_init_struct);
```

tli_init

The description of tli_init is shown as below:

Table 3-856. Function tli_init

Function name	tli_init
Function prototype	void tli_init(tli_parameter_struct *tli_struct);
Function descriptions	initialize TLI display timing parameters
Precondition	-
The called functions	-
Input parameter{in}	
*tli_struct	a pointer to tli_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
tli_parameter_struct tli_init_struct;
/* initialize the parameters of TLI parameter structure with the default values */
tli_struct_para_init(&tli_init_struct);
/* configure TLI parameter struct */
tli_init_struct.signalpolarity_hs = TLI_HSYN_ACTLIVE_LOW;
tli_init_struct.signalpolarity_vs = TLI_VSYN_ACTLIVE_LOW;
tli_init_struct.signalpolarity_de = TLI_DE_ACTLIVE_LOW;
```

```

tli_init_struct.signalpolarity_pixelck = TLI_PIXEL_CLOCK_TLI;

/* LCD display timing configuration */

tli_init_struct.synpsz_hpsz = 40;

tli_init_struct.synpsz_vpsz = 9;

tli_init_struct.backpsz_hbpsz = 42;

tli_init_struct.backpsz_vbpsz = 11;

tli_init_struct.activesz_hasz = 522;

tli_init_struct.activesz_vasz = 283;

tli_init_struct.totalsz_htsz = 524;

tli_init_struct.totalsz_vtsz = 285;

/* configure LCD background R,G,B values */

tli_init_struct.backcolor_red = 0xFF;

tli_init_struct.backcolor_green = 0xFF;

tli_init_struct.backcolor_blue = 0xFF;

tli_init(&tli_init_struct);

```

tli_dither_config

The description of tli_dither_config is shown as below:

Table 3-857. Function tli_dither_config

Function name	tli_dither_config
Function prototype	void tli_dither_config(uint8_t dither_stat);
Function descriptions	configure TLI dither function
Precondition	-
The called functions	-
Input parameter{in}	
dither_stat	dither function
<i>TLI_DITHER_ENABLE</i>	enable TLI dither function
<i>TLI_DITHER_DISABLE</i>	disable TLI dither function
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable TLI dither function */

tli_dither_config(TLI_DITHER_ENABLE);
```

tli_enable

The description of tli_enable is shown as below:

Table 3-858. Function tli_enable

Function name	tli_enable
Function prototype	void tli_enable(void);
Function descriptions	enable TLI
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enabled TLI */

tli_enable();
```

tli_disable

The description of tli_disable is shown as below:

Table 3-859. Function tli_disable

Function name	tli_disable
Function prototype	void tli_disable(void);

Function descriptions	disable TLI
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TLI */

tli_disable();
```

tli_reload_config

The description of tli_reload_config is shown as below:

Table 3-860. Function tli_reload_config

Function name	tli_reload_config
Function prototype	void tli_reload_config(uint8_t reload_mod);
Function descriptions	configure TLI reload mode
Precondition	-
The called functions	-
Input parameter{in}	
reload_mod	TLI reload mode
TLI_FRAME_BLANK_R ELOAD_EN	the layer configuration will be reloaded at frame blank
TLI_REQUEST_RELO AD_EN	the layer configuration will be reloaded after this bit sets
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure TLI reload mode */

tli_reload_config(TLI_REQUEST_RELOAD_EN);
```

tli_layer_struct_para_init

The description of tli_layer_struct_para_init is shown as below:

Table 3-861. Function tli_layer_struct_para_init

Function name	tli_layer_struct_para_init
Function prototype	void tli_layer_struct_para_init(tli_layer_parameter_struct *layer_struct);
Function descriptions	initialize the parameters of TLI layer structure with the default values, it is suggested that call this function after a tli_layer_parameter_struct structure is defined
Precondition	-
The called functions	-
Input parameter{in}	
*layer_struct	a pointer to tli_layer_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
tli_layer_parameter_struct tli_layer_init_struct;

/* initialize the parameters of TLI layer structure with the default values */

tli_layer_struct_para_init(&tli_layer_init_struct);
```

tli_layer_init

The description of tli_layer_init is shown as below:

Table 3-862. Function tli_layer_init

Function name	tli_layer_init
----------------------	----------------

Function prototype	void tli_layer_init(uint32_t layerx,tli_layer_parameter_struct *layer_struct);
Function descriptions	initialize TLI layer
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
<i>LAYERx</i>	x=0, 1
Input parameter{in}	
*layer_struct	a pointer to tli_layer_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
tli_layer_parameter_struct tli_layer_init_struct;
tli_layer_struct_para_init(&tli_layer_init_struct);
/* TLI layer0 configuration */
tli_layer_init_struct.layer_window_leftpos = 20 + 43;
tli_layer_init_struct.layer_window_rightpos = (20 + 440 + 43 - 1);
tli_layer_init_struct.layer_window_toppos = 40 + 12;
tli_layer_init_struct.layer_window_bottompos = (40 + 182 + 12 - 1);
tli_layer_init_struct.layer_ppf = LAYER_PPF_RGB565;
/* TLI window specified alpha configuration */
tli_layer_init_struct.layer_sa = 255;
/* TLI layer default alpha R,G,B value configuration */
tli_layer_init_struct.layer_default_blue = 0xFF;
tli_layer_init_struct.layer_default_green = 0xFF;
tli_layer_init_struct.layer_default_red = 0xFF;
tli_layer_init_struct.layer_default_alpha = 0xFF;
```

```

/* TLI window blend configuration */

tli_layer_init_struct.layer_acf1 = LAYER_ACF1_PASA;
tli_layer_init_struct.layer_acf2 = LAYER_ACF2_PASA;

/* TLI layer frame buffer base address configuration */

tli_layer_init_struct.layer_frame_bufaddr = (uint32_t)&gBackground;
tli_layer_init_struct.layer_frame_line_length = ((440 * 2) + 3);
tli_layer_init_struct.layer_frame_buf_stride_offset = (440 * 2);
tli_layer_init_struct.layer_frame_total_line_number = 182;
tli_layer_init(LAYER0, &tli_layer_init_struct);

```

tli_layer_window_offset_modify

The description of tli_layer_window_offset_modify is shown as below:

Table 3-863. Function tli_layer_window_offset_modify

Function name	tli_layer_window_offset_modify
Function prototype	void tli_layer_window_offset_modify(uint32_t layerx,uint16_t offset_x,uint16_t offset_y);
Function descriptions	reconfigure window position
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
<i>LAYERx</i>	x=0, 1
Input parameter{in}	
offset_x	new horizontal offset
Input parameter{in}	
offset_y	new vertical offset
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reconfigure LAYER1 window position */

tli_layer_window_offset_modify(LAYER1, 20, 20);
```

tli_lut_struct_para_init

The description of tli_lut_struct_para_init is shown as below:

Table 3-864. Function tli_lut_struct_para_init

Function name	tli_lut_struct_para_init
Function prototype	void tli_lut_struct_para_init(tli_layer_lut_parameter_struct *lut_struct);
Function descriptions	initialize the parameters of TLI layer LUT structure with the default values, it is suggested that call this function after a tli_layer_lut_parameter_struct structure is defined
Precondition	-
The called functions	-
Input parameter{in}	
*lut_struct	a pointer to tli_layer_lut_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
tli_layer_lut_parameter_struct tli_lut_struct;
/* initialize the parameters of TLI layer LUT structure with the default values */
tli_lut_struct_para_init(&tli_lut_struct);
```

tli_lut_init

The description of tli_lut_init is shown as below:

Table 3-865. Function tli_lut_init

Function name	tli_lut_init
Function prototype	void tli_lut_init(uint32_t layerx,tli_layer_lut_parameter_struct *lut_struct);
Function descriptions	initialize TLI layer LUT

Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
LAYERx	x=0, 1
Input parameter{in}	
*lut_struct	a pointer to tli_layer_lut_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
tli_layer_lut_parameter_struct tli_lut_struct;
tli_lut_struct_para_init(&tli_lut_struct);
/* initialize TLI layer0 LUT */
tli_lut_struct.layer_table_addr = 0x20003000;
tli_lut_struct.layer_lut_channel_red = 0x20;
tli_lut_struct.layer_lut_channel_green = 0x30;
tli_lut_struct.layer_lut_channel_blue = 0xFF;
tli_lut_init(LAYER0, &tli_lut_struct);
```

tli_color_key_init

The description of tli_color_key_init is shown as below:

Table 3-866. Function tli_color_key_init

Function name	tli_color_key_init
Function prototype	void tli_color_key_init(uint32_t layerx,uint32_t redkey,uint32_t greenkey,uint32_t bluekey);
Function descriptions	initialize TLI layer color key
Precondition	-

The called functions	-
Input parameter{in}	
layerx	layer
LAYERx	x=0, 1
Input parameter{in}	
redkey	color key red
Input parameter{in}	
greenkey	color key green
Input parameter{in}	
bluekey	color key blue
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TLI layer0 color key */
tli_color_key_init(LAYER0, 0xAA, 0xFF, 0x00);
```

tli_layer_enable

The description of tli_layer_enable is shown as below:

Table 3-867. Function tli_layer_enable

Function name	tli_layer_enable
Function prototype	void tli_layer_enable(uint32_t layerx);
Function descriptions	enable TLI layer
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
LAYERx	x=0, 1

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TLI layer0 */

tli_layer_enable(LAYER0);
```

tli_layer_disable

The description of tli_layer_disable is shown as below:

Table 3-868. Function tli_layer_disable

Function name	tli_layer_disable
Function prototype	void tli_layer_disable(uint32_t layerx);
Function descriptions	disable TLI layer
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
LAYERx	x=0, 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TLI layer0 */

tli_layer_disable(LAYER0);
```

tli_color_key_enable

The description of tli_color_key_enable is shown as below:

Table 3-869. Function tli_color_key_enable

Function name	tli_color_key_enable
Function prototype	void tli_color_key_enable(uint32_t layerx);
Function descriptions	enable TLI layer color keying
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
<i>LAYERx</i>	x=0, 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TLI layer0 color keying */
tli_color_key_enable(LAYER0);
```

tli_color_key_disable

The description of tli_color_key_disable is shown as below:

Table 3-870. Function tli_color_key_disable

Function name	tli_color_key_disable
Function prototype	void tli_color_key_disable(uint32_t layerx);
Function descriptions	disable TLI layer color keying
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
<i>LAYERx</i>	x=0, 1
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable TLI layer0 color keying */
tli_color_key_disable(LAYER0);
```

tli_lut_enable

The description of tli_lut_enable is shown as below:

Table 3-871. Function tli_lut_enable

Function name	tli_lut_enable
Function prototype	void tli_lut_enable(uint32_t layerx);
Function descriptions	enable TLI layer LUT
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
LAYERx	x=0, 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TLI layer0 LUT */
tli_lut_enable(LAYER0);
```

tli_lut_disable

The description of tli_lut_disable is shown as below:

Table 3-872. Function tli_lut_disable

Function name	tli_lut_disable
----------------------	-----------------

Function prototype	void tli_lut_disable(uint32_t layerx);
Function descriptions	disable TLI layer0 LUT
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
LAYERx	x=0, 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TLI layer0 LUT */
tli_lut_disable(LAYER0);
```

tli_line_mark_set

The description of tli_line_mark_set is shown as below:

Table 3-873. Function tli_line_mark_set

Function name	tli_line_mark_set
Function prototype	void tli_line_mark_set(uint32_t line_num);
Function descriptions	set line mark value
Precondition	-
The called functions	-
Input parameter{in}	
line_num	line number
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set line mark value */

tli_line_mark_set(0x20);
```

tli_current_pos_get

The description of tli_current_pos_get is shown as below:

Table 3-874. Function tli_current_pos_get

Function name	tli_current_pos_get
Function prototype	uint32_t tli_current_pos_get(void);
Function descriptions	get current displayed position
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	position of current pixel

Example:

```
uint32_t pos;

/* get current pixel position */

pos = tli_current_pos_get();
```

tli_interrupt_enable

The description of tli_interrupt_enable is shown as below:

Table 3-875. Function tli_interrupt_enable

Function name	tli_interrupt_enable
Function prototype	void tli_interrupt_enable(uint32_t int_flag);
Function descriptions	enable TLI interrupt
Precondition	-

The called functions	-
Input parameter{in}	
int_flag	TLI interrupt flags
<i>TLI_INT_LM</i>	line mark interrupt
<i>TLI_INT_FE</i>	FIFO error interrupt
<i>TLI_INT_TE</i>	transaction error interrupt
<i>TLI_INT_LCR</i>	layer configuration reloaded interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TLI line mark interrupt */
tli_interrupt_enable(TLI_INT_LM);
```

tli_interrupt_disable

The description of tli_interrupt_disable is shown as below:

Table 3-876. Function tli_interrupt_disable

Function name	tli_interrupt_disable
Function prototype	void tli_interrupt_disable(uint32_t int_flag);
Function descriptions	disable TLI interrupt
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	TLI interrupt flags
<i>TLI_INT_LM</i>	line mark interrupt
<i>TLI_INT_FE</i>	FIFO error interrupt
<i>TLI_INT_TE</i>	transaction error interrupt
<i>TLI_INT_LCR</i>	layer configuration reloaded interrupt

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TLI line mark interrupt */

tli_interrupt_disable(TLI_INT_LM);
```

tli_interrupt_flag_get

The description of tli_interrupt_flag_get is shown as below:

Table 3-877. Function tli_interrupt_flag_get

Function name	tli_interrupt_flag_get
Function prototype	FlagStatus tli_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get TLI interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	TLI interrupt flags
<i>TLI_INT_FLAG_LM</i>	line mark interrupt flag
<i>TLI_INT_FLAG_FE</i>	FIFO error interrupt flag
<i>TLI_INT_FLAG_TE</i>	transaction error interrupt flag
<i>TLI_INT_FLAG_LCR</i>	layer configuration reloaded interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get TLI interrupt flag */

if(SET == tli_interrupt_flag_get(TLI_INT_FLAG_LM)){
```

```

    tli_interrupt_flag_clear(TLI_INT_FLAG_LM);
}

}

```

tli_interrupt_flag_clear

The description of tli_interrupt_flag_clear is shown as below:

Table 3-878. Function tli_interrupt_flag_clear

Function name	tli_interrupt_flag_clear
Function prototype	void tli_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear TLI interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
<i>int_flag</i>	TLI interrupt flags
<i>TLI_INT_FLAG_LM</i>	line mark interrupt flag
<i>TLI_INT_FLAG_FE</i>	FIFO error interrupt flag
<i>TLI_INT_FLAG_TE</i>	transaction error interrupt flag
<i>TLI_INT_FLAG_LCR</i>	layer configuration reloaded interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

if(SET == tli_interrupt_flag_get(TLI_INT_FLAG_LM)){
    /* clear TLI interrupt flag */
    tli_interrupt_flag_clear(TLI_INT_FLAG_LM);
}

```

tli_flag_get

The description of tli_flag_get is shown as below:

Table 3-879. Function tli_flag_get

Function name	tli_flag_get
Function prototype	FlagStatus tli_flag_get(uint32_t flag);
Function descriptions	get TLI flag or state in TLI_INTF register or TLI_STAT register
Precondition	-
The called functions	-
Input parameter{in}	
flag	TLI flags or states
<i>TLI_FLAG_VDE</i>	current VDE state
<i>TLI_FLAG_HDE</i>	current HDE state
<i>TLI_FLAG_VS</i>	current VS status of the TLI
<i>TLI_FLAG_HS</i>	current HS status of the TLI
<i>TLI_FLAG_LM</i>	line mark interrupt flag
<i>TLI_FLAG_FE</i>	FIFO error interrupt flag
<i>TLI_FLAG_TE</i>	transaction error interrupt flag
<i>TLI_FLAG_LCR</i>	layer configuration reloaded interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```

/* wait the TLI_FLAG_LM flag set */

while(RESET == tli_flag_get(TLI_FLAG_LM)){
}

```

3.28. TRNG

The true random number generator (TRNG) can generate a 32-bit random value by using continuous analog noise. The TRNG registers are listed in chapter [3.28.1](#), the TRNG firmware functions are introduced in chapter [3.28.2](#).

3.28.1. Descriptions of Peripheral registers

TRNG registers are listed in the table shown as below:

Table 3-880 TRNG Registers

Registers	Descriptions
TRNG_CTL	TRNG control register
TRNG_STAT	TRNG status register
TRNG_DATA	TRNG data register

3.28.2. Descriptions of Peripheral functions

TRNG firmware functions are listed in the table shown as below:

Table 3-881. TRNG firmware function

Function name	Function description
trng_deinit	deinitialize the TRNG
trng_enable	enable the TRNG interface
trng_disable	disable the TRNG interface
trng_get_true_random_data	get the true random data
trng_interrupt_enable	enable the TRNG interrupt
trng_interrupt_disable	disable the TRNG interrupt
trng_flag_get	get the TRNG status flags
trng_interrupt_flag_get	get the TRNG interrupt flags
trng_interrupt_flag_clear	clear the TRNG interrupt flags

trng_deinit

The description of trng_deinit is shown as below:

Table 3-882. Function trng_deinit

Function name	trng_deinit
Function prototype	void trng_deinit(void)
Function descriptions	reset TRNG peripheral
Precondition	-

The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset TRNG */
trng_deinit();
```

trng_enable

The description of trng_enable is shown as below:

Table 3-883. Function trng_enable

Function name	trng_enable
Function prototype	void trng_enable(void)
Function descriptions	enable the TRNG interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TRNG */
trng_enable();
```

trng_disable

The description of trng_disable is shown as below:

Table 3-884 Function trng_disable

Function name	trng_disable
Function prototype	void trng_disable(void)
Function descriptions	disable the TRNG interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TRNG */
trng_disable();
```

trng_get_true_random_data

The description of trng_get_true_random_data is shown as below:

Table 3-885 Function trng_get_true_random_data

Function name	trng_get_true_random_data
Function prototype	uint32_t trng_get_true_random_data(void)
Function descriptions	get the true random data
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

Return value	
uint32_t	the generated random data

Example:

```
/* get true random data */

uint32_t data;

data = trng_get_true_random_data();
```

trng_interrupt_enable

The description of trng_interrupt_enable is shown as below:

Table 3-886 Function trng_interrupt_enable

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TRNG interrupt */

trng_interrupt_enable();
```

trng_interrupt_disable

The description of trng_interrupt_disable is shown as below:

Table 3-887 Function trng_interrupt_disable

Function name	

Function prototype	void trng_interrupt_disable(void)
Function descriptions	disable the TRNG interrupt
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TRNG interrupt */

trng_interrupt_disable();
```

trng_flag_get

The description of trng_flag_get is shown as below:

Table 3-888 Function trng_flag_get

Function name	trng_flag_get
Function prototype	FlagStatus trng_flag_get(trng_flag_enum flag)
Function descriptions	get the trng status flags
Precondition	-
The called functions	-
Input parameter{in}	
flag	trng status flag
<i>TRNG_FLAG_DRDY</i>	Random Data ready status
<i>TRNG_FLAG_CECS</i>	Clock error current status
<i>TRNG_FLAG_SECS</i>	Seed error current status
Output parameter{out}	
-	-

Return value	
FlagStatus	SET or RESET

Example:

```
/* get TRNG clock error current flag status*/
FlagStatus flag_status = RESET;
flag_status == trng_flag_get(TRNG_FLAG_CECS);
```

trng_interrupt_flag_get

The description of trng_interrupt_flag_get is shown as below:

Table 3-889 Function trng_interrupt_flag_get

Function name	trng_interrupt_flag_get
Function prototype	FlagStatus trng_interrupt_flag_get(trng_int_flag_enum int_flag)
Function descriptions	get the trng interrupt flags
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	trng interrupt flag
<i>TRNG_INT_FLAG_CEI</i> <i>F</i>	clock error interrupt flag
<i>TRNG_INT_FLAG_SEI</i> <i>F</i>	Seed error interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get TRNG clock error interrupt flag*/
FlagStatus interrupt_flag = RESET;
interrupt_flag = trng_interrupt_flag_get(TRNG_INT_FLAG_CEIF);
```

trng_interrupt_flag_clear

The description of trng_interrupt_flag_clear is shown as below:

Table 3-890 Function trng_interrupt_flag_clear

Function name	trng_interrupt_flag_clear
Function prototype	void trng_interrupt_flag_get(trng_int_flag_enum int_flag)
Function descriptions	clear the trng interrupt flags
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	trng interrupt flag
<i>TRNG_INT_FLAG_CEIF</i>	clock error interrupt flag
<i>TRNG_INT_FLAG_SEIF</i>	Seed error interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TRNG clock error interrupt flag*/
trng_interrupt_flag_clear(TRNG_INT_FLAG_CEIF);
```

3.29. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.29.1](#), the USART firmware functions are introduced in chapter [3.29.2](#).

3.29.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

Table 3-891. USART Registers

Registers	Descriptions
USART_STAT0	Status register 0
USART_DATA	Data register
USART_BAUD	Baud rate register
USART_CTL0	Control register 0
USART_CTL1	Control register 1
USART_CTL2	Control register 2
USART_GP	Guard time and prescaler register
USART_CTL3	Control register 3
USART_RT	Receiver timeout register
USART_STAT1	Status register 1
USART_CHC	Coherence control register

3.29.2. Descriptions of Peripheral functions

USART firmware functions are listed in the table shown as below:

Table 3-892. USART firmware function

Function name	Function description
uart_deinit	reset USART/UART
uart_baudrate_set	configure USART baud rate value
uart_parity_config	configure USART parity
uart_word_length_set	configure USART word length
uart_stop_bit_set	configure USART stop bit length
uart_enable	enable USART
uart_disable	disable USART
uart_transmit_config	configure USART transmitter
uart_receive_config	configure USART receiver
uart_data_first_config	data is transmitted/received with the LSB/MSB first
uart_invert_config	configure USART inverted
uart_oversample_config	configure the USART oversample mode
uart_sample_bit_config	configure sample bit method
uart_receiver_timeout_enable	enable receiver timeout
uart_receiver_timeout_disable	disable receiver timeout
uart_receiver_timeout_threshold_config	configure receiver timeout threshold
uart_data_transmit	USART transmit data function
uart_data_receive	USART receive data function
uart_address_config	configure the address of the USART in wake up by address match mode
uart_mute_mode_enable	enable mute mode
uart_mute_mode_disable	disable mute mode

Function name	Function description
uart_mute_mode_wakeup_config	configure wakeup method in mute mode
uart_lin_mode_enable	enable LIN mode
uart_lin_mode_disable	disable LIN mode
uart_lin_break_dection_length_config	configure LIN break frame length
uart_send_break	send break frame
uart_halfduplex_enable	enable half duplex mode
uart_halfduplex_disable	disable half duplex mode
uart_synchronous_clock_enable	enable CK pin in synchronous mode
uart_synchronous_clock_disable	disable CK pin in synchronous mode
uart_synchronous_clock_config	configure USART synchronous mode parameters
uart_guard_time_config	configure guard time value in smartcard mode
uart_smartcard_mode_enable	enable smartcard mode
uart_smartcard_mode_disable	disable smartcard mode
uart_smartcard_mode_nack_enable	enable NACK in smartcard mode
uart_smartcard_mode_nack_disable	disable NACK in smartcard mode
uart_smartcard_autoretry_config	configure smartcard auto-retry number
uart_block_length_config	configure block length
uart_irda_mode_enable	enable IrDA mode
uart_irda_mode_disable	disable IrDA mode
uart_prescaler_config	configure the peripheral clock prescaler in USART IrDA low-power mode
uart_irda_lowpower_config	configure IrDA low-power
uart_hardware_flow_rts_config	configure hardware flow control RTS
uart_hardware_flow_cts_config	configure hardware flow control CTS
uart_break_frame_coherence_config	configure break frame coherence mode
uart_parity_check_coherence_config	configure parity check coherence mode
uart_hardware_flow_coherence_config	configure hardware flow control coherence mode
uart_dma_receive_config	configure USART DMA reception
uart_dma_transmit_config	configure USART DMA transmission
uart_flag_get	get flag in STAT0/STAT1 register
uart_flag_clear	clear flag in STAT0/STAT1 register
uart_interrupt_enable	enable USART interrupt
uart_interrupt_disable	disable USART interrupt
uart_interrupt_flag_get	get USART interrupt flag status
uart_interrupt_flag_clear	clear USART interrupt flag

uart_deinit

The description of `uart_deinit` is shown as below:

Table 3-893. Function usart_deinit

Function name	usart_deinit
Function prototype	void usart_deinit(uint32_t usart_periph);
Function descriptions	reset USART/UART
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset USART0 */

usart_deinit (USART0);
```

usart_baudrate_set

The description of usart_baudrate_set is shown as below:

Table 3-894. Function usart_baudrate_set

Function name	usart_baudrate_set
Function prototype	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
Function descriptions	configure USART baud rate value
Precondition	-
The called functions	rcu_clock_freq_get
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
baudval	baud rate value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 baud rate value */
```

```
uart_baudrate_set(USART0, 115200);
```

uart_parity_config

The description of `uart_parity_config` is shown as below:

Table 3-895. Function `uart_parity_config`

Function name	uart_parity_config
Function prototype	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
Function descriptions	configure USART parity
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
paritycfg	configure USART parity
USART_PM_NONE	no parity
USART_PM_ODD	odd parity
USART_PM EVEN	even parity
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART parity */

uart_parity_config(USART0, USART_PM_EVEN);
```

uart_word_length_set

The description of `uart_word_length_set` is shown as below:

Table 3-896. Function `uart_word_length_set`

Function name	uart_word_length_set
Function prototype	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
Function descriptions	configure USART word length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7

Input parameter{in}	
wlen	USART word length
<i>USART_WL_8BIT</i>	8 bits
<i>USART_WL_9BIT</i>	9 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 word length */

uart_word_length_set(USART0, USART_WL_9BIT);
```

uart_stop_bit_set

The description of `uart_stop_bit_set` is shown as below:

Table 3-897. Function `uart_stop_bit_set`

Function name	uart_stop_bit_set
Function prototype	void uart_stop_bit_set(uint32_t usart_periph, uint32_t strlen);
Function descriptions	configure USART stop bit length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
strlen	USART stop bit
<i>USART_STB_1BIT</i>	1 bit
<i>USART_STB_0_5BIT</i>	0.5 bit, not available for UARTx(x=3,4,6,7)
<i>USART_STB_2BIT</i>	2 bits
<i>USART_STB_1_5BIT</i>	1.5 bits, not available for USARTx(x=3,4,6,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 stop bit length */

uart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

uart_enable

The description of `uart_enable` is shown as below:

Table 3-898. Function `uart_enable`

Function name	uart_enable
Function prototype	<code>void usart_enable(uint32_t usart_periph);</code>
Function descriptions	enable USART
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	x=0,1,2,5
<code>UARTx</code>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 */
usart_enable(USART0);
```

uart_disable

The description of `uart_disable` is shown as below:

Table 3-899. Function `uart_disable`

Function name	uart_disable
Function prototype	<code>void usart_disable(uint32_t usart_periph);</code>
Function descriptions	disable USART
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	x=0,1,2,5
<code>UARTx</code>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 */
```

```
uart_disable(USART0);
```

uart_transmit_config

The description of `uart_transmit_config` is shown as below:

Table 3-900. Function `uart_transmit_config`

Function name	uart_transmit_config
Function prototype	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
Function descriptions	configure USART transmitter
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
txconfig	enable or disable USART transmitter
USART_TRANSMIT_ENABLE	enable USART transmission
USART_TRANSMIT_DISABLE	disable USART transmission
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 transmitter */
usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

uart_receive_config

The description of `uart_receive_config` is shown as below:

Table 3-901. Function `uart_receive_config`

Function name	uart_receive_config
Function prototype	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
Function descriptions	configure USART receiver
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5

UARTx	x=3,4,6,7
Input parameter{in}	
rxconfig	enable or disable USART receiver
USART_RECEIVE_ENABLE	enable USART reception
USART_RECEIVE_DISABLE	disable USART reception
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 receiver */

uart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

uart_data_first_config

The description of `uart_data_first_config` is shown as below:

Table 3-902. Function `uart_data_first_config`

Function name	uart_data_first_config
Function prototype	void <code>uart_data_first_config(uint32_t usart_periph, uint32_t msbf);</code>
Function descriptions	data is transmitted/received with the LSB/MSB first
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Input parameter{in}	
msbf	LSB first or MSB first
USART_MSBF_LSB	LSB first
USART_MSBF_MSB	MSB first
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure LSB of data first */

uart_data_first_config(USART0, USART_MSBF_LSB);
```

uart_invert_config

The description of `uart_invert_config` is shown as below:

Table 3-903. Function `uart_invert_config`

Function name	<code>uart_invert_config</code>
Function prototype	<code>void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);</code>
Function descriptions	configure USART inversion
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Input parameter{in}	
invertpara	refer to enum <code>usart_invert_enum</code>
USART_DINV_ENABL_E	data bit level inversion
USART_DINV_DISABL_E	data bit level not inversion
USART_TXPIN_ENABL_E	TX pin level inversion
USART_TXPIN_DISABL_E	TX pin level not inversion
USART_RXPIN_ENABL_E	RX pin level inversion
USART_RXPIN_DISABL_E	RX pin level not inversion
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART inversion */

uart_invert_config(USART0, USART_DINV_ENABLE);
```

uart_oversample_config

The description of `uart_oversample_config` is shown as below:

Table 3-904. Function `uart_oversample_config`

Function name	<code>uart_oversample_config</code>
Function prototype	<code>void usart_oversample_config(uint32_t usart_periph, uint32_t oversamp);</code>

Function descriptions	configure the USART oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
uart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
oversamp	oversample value
USART_OVSMOD_8	8 bits
USART_OVSMOD_16	16 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 oversample mode */
uart_oversample_config(USART0, USART_OVSMOD_8);
```

uart_sample_bit_config

The description of `uart_sample_bit_config` is shown as below:

Table 3-905. Function `uart_sample_bit_config`

Function name	<code>uart_sample_bit_config</code>
Function prototype	<code>void usart_sample_bit_config(uint32_t usart_periph, uint32_t obsm);</code>
Function descriptions	configure sample bit method
Precondition	-
The called functions	-
Input parameter{in}	
uart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
obsm	sample bit
USART_OSB_1bit	1 bits
USART_OSB_3bit	3 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure USART0 sample bit */

uart_sample_bit_config(USART0, USART_OSB_1bit);

```

uart_receiver_timeout_enable

The description of `uart_receiver_timeout_enable` is shown as below:

Table 3-906. Function `uart_receiver_timeout_enable`

Function name	uart_receiver_timeout_enable
Function prototype	void uart_receiver_timeout_enable(uint32_t usart_periph);
Function descriptions	enable receiver timeout
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable receiver timeout of USART */

uart_receiver_timeout_enable(USART0);

```

uart_receiver_timeout_disable

The description of `uart_receiver_timeout_disable` is shown as below:

Table 3-907. Function `uart_receiver_timeout_disable`

Function name	uart_receiver_timeout_disable
Function prototype	void uart_receiver_timeout_disable(uint32_t usart_periph);
Function descriptions	disable receiver timeout
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable receiver timeout of USART */
uart_receiver_timeout_disable(USART0);
```

uart_receiver_timeout_threshold_config

The description of `uart_receiver_timeout_threshold_config` is shown as below:

Table 3-908. Function `uart_receiver_timeout_threshold_config`

Function name	uart_receiver_timeout_threshold_config
Function prototype	void <code>uart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);</code>
Function descriptions	configure receiver timeout threshold
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Input parameter{in}	
rtimeout	timeout value
0x0xFFFFFFF	timeout value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the receiver timeout threshold of USART0 */
uart_receiver_timeout_threshold_config(USART0, 115200*3);
```

uart_data_transmit

The description of `uart_data_transmit` is shown as below:

Table 3-909. Function `uart_data_transmit`

Function name	uart_data_transmit
Function prototype	void <code>uart_data_transmit(uint32_t usart_periph, uint32_t data);</code>
Function descriptions	USART transmit data function
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5

<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
data	data of transmission
0-0x1FF	data of transmission
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 transmit data */

uart_data_transmit(USART0, 0xAA);
```

uart_data_receive

The description of `uart_data_receive` is shown as below:

Table 3-910. Function `uart_data_receive`

Function name	uart_data_receive
Function prototype	void usart_receive_config(uint32_t usart_periph);
Function descriptions	USART receive data function
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
uint32_t	data of received(0-0x1FF)

Example:

```
/* USART0 receive data */

uint16_t temp;

temp = usart_data_receive(USART0);
```

uart_address_config

The description of `uart_address_config` is shown as below:

Table 3-911. Function `uart_address_config`

Function name	uart_address_config
----------------------	---------------------

Function prototype	void usart_address_config(uint32_t usart_periph, uint8_t addr);
Function descriptions	configure the address of the USART in wake up by address match mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
addr	address of USART/UART
<i>0-0xFF</i>	address of USART/UART
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure address of the USART0 */
usart_address_config(USART0, 0x00);
```

usart_mute_mode_enable

The description of `usart_mute_mode_enable` is shown as below:

Table 3-912. Function usart_mute_mode_enable

Function name	usart_mute_mode_enable
Function prototype	void usart_mute_mode_enable(uint32_t usart_periph);
Function descriptions	enable mute mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 receiver in mute mode */
usart_mute_mode_enable(USART0);
```

uart_mute_mode_disable

The description of `uart_mute_mode_disable` is shown as below:

Table 3-913. Function `uart_mute_mode_disable`

Function name	uart_mute_mode_disable
Function prototype	void <code>uart_mute_mode_disable(uint32_t usart_periph);</code>
Function descriptions	disable mute mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	x=0,1,2,5
<code>UARTx</code>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 receiver in mute mode */

uart_mute_mode_disable(USART0);
```

uart_mute_mode_wakeup_config

The description of `uart_mute_mode_wakeup_config` is shown as below:

Table 3-914. Function `uart_mute_mode_wakeup_config`

Function name	uart_mute_mode_wakeup_config
Function prototype	void <code>uart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);</code>
Function descriptions	configure wakeup method in mute mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	x=0,1,2,5
<code>UARTx</code>	x=3,4,6,7
Input parameter{in}	
<code>wmethod</code>	two methods be used to enter or exit the mute mode
<code>USART_WM_IDLE</code>	idle line
<code>USART_WM_ADDR</code>	address mask
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure USART0 wakeup method in mute mode */
uart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

uart_lin_mode_enable

The description of `uart_lin_mode_enable` is shown as below:

Table 3-915. Function `uart_lin_mode_enable`

Function name	uart_lin_mode_enable
Function prototype	void usart_lin_mode_enable(uint32_t usart_periph);
Function descriptions	enable LIN mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 LIN mode enable */
uart_lin_mode_enable(USART0);
```

uart_lin_mode_disable

The description of `uart_lin_mode_disable` is shown as below:

Table 3-916. Function `uart_lin_mode_disable`

Function name	uart_lin_mode_disable
Function prototype	void usart_lin_mode_disable(uint32_t usart_periph);
Function descriptions	disable LIN mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5

<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 LIN mode disable */

uart_lin_mode_disable(USART0);
```

uart_lin_break_dection_length_config

The description of `uart_lin_break_dection_length_config` is shown as below:

Table 3-917. Function `uart_lin_break_dection_length_config`

Function name	uart_lin_break_dection_length_config
Function prototype	void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t Iblen);
Function descriptions	configure LIN break frame length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
Iblen	two methods be used to enter or exit the mute mode
<i>USART_LBLEN_10B</i>	break frame length is 10 bits
<i>USART_LBLEN_11B</i>	break frame length is 11 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure LIN break frame length */

uart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

uart_send_break

The description of `uart_send_break` is shown as below:

Table 3-918. Function usart_send_break

Function name	usart_send_break
Function prototype	void usart_send_break(uint32_t usart_periph);
Function descriptions	send break frame
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 send break frame */

usart_send_break(USART0);
```

usart_halfduplex_enable

The description of usart_halfduplex_enable is shown as below:

Table 3-919. Function usart_halfduplex_enable

Function name	usart_halfduplex_enable
Function prototype	void usart_halfduplex_enable(uint32_t usart_periph);
Function descriptions	enable half duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 half duplex mode*/

usart_halfduplex_enable(USART0);
```

uart_halfduplex_disable

The description of `uart_halfduplex_disable` is shown as below:

Table 3-920. Function `uart_halfduplex_disable`

Function name	uart_halfduplex_disable
Function prototype	<code>void usart_halfduplex_disable(uint32_t usart_periph);</code>
Function descriptions	disable half duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 half duplex mode*/
usart_halfduplex_disable(USART0);
```

uart_synchronous_clock_enable

The description of `uart_synchronous_clock_enable` is shown as below:

Table 3-921. Function `uart_synchronous_clock_enable`

Function name	uart_synchronous_clock_enable
Function prototype	<code>void usart_synchronous_clock_enable(uint32_t usart_periph);</code>
Function descriptions	enable CK pin in synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 CK pin in synchronous mode */
```

```
uart_synchronous_clock_enable(USART0);
```

uart_synchronous_clock_disable

The description of `uart_synchronous_clock_disable` is shown as below:

Table 3-922. Function `uart_synchronous_clock_disable`

Function name	uart_synchronous_clock_disable
Function prototype	void uart_synchronous_clock_disable(uint32_t usart_periph);
Function descriptions	disable CK pin in synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 CK pin in synchronous mode */

uart_synchronous_clock_disable(USART0);
```

uart_synchronous_clock_config

The description of `uart_synchronous_clock_config` is shown as below:

Table 3-923. Function `uart_synchronous_clock_config`

Function name	uart_synchronous_clock_config
Function prototype	void uart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
Function descriptions	configure USART synchronous mode parameters
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Input parameter{in}	
clen	CK length
USART_CLEN_NONE	there are 7 CK pulses for an 8 bit frame and 8 CK pulses for a 9 bit frame
USART_CLEN_EN	there are 8 CK pulses for an 8 bit frame and 9 CK pulses for a 9 bit frame
Input parameter{in}	
cph	clock phase

<i>USART_CPH_1CK</i>	first clock transition is the first data capture edge
<i>USART_CPH_2CK</i>	second clock transition is the first data capture edge
Input parameter{in}	
<i>cpl</i>	clock polarity
<i>USART_CPL_LOW</i>	steady low value on CK pin
<i>USART_CPL_HIGH</i>	steady high value on CK pin
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure USART0 synchronous mode parameters */

uart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,
USART_CPL_HIGH);

```

uart_guard_time_config

The description of `uart_guard_time_config` is shown as below:

Table 3-924. Function `uart_guard_time_config`

Function name	<code>uart_guard_time_config</code>
Function prototype	<code>void uart_guard_time_config(uint32_t usart_periph,uint32_t gaut);</code>
Function descriptions	configure guard time value in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Input parameter{in}	
gaut	guard time value
0-0x000000FF	guard time value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure USART0 guard time value in smartcard mode */

uart_guard_time_config(USART0, 0x0000 0055);

```

uart_smartcard_mode_enable

The description of `uart_smartcard_mode_enable` is shown as below:

Table 3-925. Function `uart_smartcard_mode_enable`

Function name	uart_smartcard_mode_enable
Function prototype	<code>void usart_smartcard_mode_enable(uint32_t usart_periph);</code>
Function descriptions	enable smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 smartcard mode enable */
uart_smartcard_mode_enable(USART0);
```

uart_smartcard_mode_disable

The description of `uart_smartcard_mode_disable` is shown as below:

Table 3-926. Function `uart_smartcard_mode_disable`

Function name	uart_smartcard_mode_disable
Function prototype	<code>void usart_smartcard_mode_disable(uint32_t usart_periph);</code>
Function descriptions	disable smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 smartcard mode disable */
uart_smartcard_mode_disable(USART0);
```

uart_smartcard_mode_nack_enable

The description of `uart_smartcard_mode_nack_enable` is shown as below:

Table 3-927. Function `uart_smartcard_mode_nack_enable`

Function name	uart_smartcard_mode_nack_enable
Function prototype	void uart_smartcard_mode_nack_enable(uint32_t usart_periph);
Function descriptions	enable NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 NACK in smartcard mode */
uart_smartcard_mode_nack_enable(USART0);
```

uart_smartcard_mode_nack_disable

The description of `uart_smartcard_mode_nack_disable` is shown as below:

Table 3-928. Function `uart_smartcard_mode_nack_disable`

Function name	uart_smartcard_mode_nack_disable
Function prototype	void uart_smartcard_mode_nack_disable(uint32_t usart_periph);
Function descriptions	disable NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 NACK in smartcard mode */
uart_smartcard_mode_nack_disable(USART0);
```

uart_smartcard_autoretry_config

The description of `uart_smartcard_autoretry_config` is shown as below:

Table 3-929. Function `uart_smartcard_autoretry_config`

Function name	uart_smartcard_autoretry_config
Function prototype	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrnum);
Function descriptions	configure smartcard auto-retry number
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Input parameter{in}	
scrnum	smartcard auto-retry number
0-0xFFFFFFFF	smartcard auto-retry number
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure smartcard auto-retry number */

uart_smartcard_autoretry_config (USART0, 0xFFFFFFFF);
```

uart_block_length_config

The description of `uart_block_length_config` is shown as below:

Table 3-930. Function `uart_block_length_config`

Function name	uart_block_length_config
Function prototype	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
Function descriptions	configure block length in Smartcard T=1 reception
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Input parameter{in}	
bl	block length
0-0xFFFFFFFF	block length
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure block length in Smartcard T=1 reception */
uart_block_length_config(USART0, 0xFFFFFFFF);
```

uart_irda_mode_enable

The description of `uart_irda_mode_enable` is shown as below:

Table 3-931. Function `uart_irda_mode_enable`

Function name	uart_irda_mode_enable
Function prototype	void uart_irda_mode_enable(uint32_t usart_periph);
Function descriptions	enable IrDA mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 IrDA mode */
uart_irda_mode_enable(USART0);
```

uart_irda_mode_disable

The description of `uart_irda_mode_disable` is shown as below:

Table 3-932. Function `uart_irda_mode_disable`

Function name	uart_irda_mode_disable
Function prototype	void uart_irda_mode_disable(uint32_t usart_periph);
Function descriptions	disable IrDA mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5

UARTx	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 IrDA mode */

uart_irda_mode_disable(USART0);
```

uart_prescaler_config

The description of **uart_prescaler_config** is shown as below:

Table 3-933. Function **uart_prescaler_config**

Function name	uart_prescaler_config
Function prototype	void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);
Function descriptions	configure the peripheral clock prescaler in USART IrDA low-power mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
psc	0x00-0xFF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the USART0 peripheral clock prescaler */

uart_prescaler_config(USART0, 0x00);
```

uart_irda_lowpower_config

The description of **uart_irda_lowpower_config** is shown as below:

Table 3-934. Function **uart_irda_lowpower_config**

Function name	uart_irda_lowpower_config
Function prototype	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
Function descriptions	configure IrDA low-power

Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
irlp	IrDA low-power or normal
<i>USART_IRLP_LOW</i>	low-power
<i>USART_IRLP_NORMA L</i>	normal
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 IrDA low-power */
uart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

uart_hardware_flow_rts_config

The description of `uart_hardware_flow_rts_config` is shown as below:

Table 3-935. Function `uart_hardware_flow_rts_config`

Function name	uart_hardware_flow_rts_config
Function prototype	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
Function descriptions	configure hardware flow control RTS
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
rtsconfig	enable or disable RTS
<i>USART_RTS_ENABLE</i>	enable RTS
<i>USART_RTS_DISABLE</i>	disable RTS
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure USART0 hardware flow control RTS */

uart_hardware_flow_cts_config(USART0, USART_RTS_ENABLE);

```

uart_hardware_flow_cts_config

The description of `uart_hardware_flow_cts_config` is shown as below:

Table 3-936. Function `uart_hardware_flow_cts_config`

Function name	uart_hardware_flow_cts_config
Function prototype	void uart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
Function descriptions	configure hardware flow control RTS
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Input parameter{in}	
ctsconfig	enable or disable CTS
USART_CTS_ENABLE	enable CTS
USART_CTS_DISABLE	disable CTS
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure USART0 hardware flow control CTS */

uart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);

```

uart_break_frame_coherence_config

The description of `uart_break_frame_coherence_config` is shown as below:

Table 3-937. Function `uart_break_frame_coherence_config`

Function name	uart_break_frame_coherence_config
Function prototype	void uart_break_frame_coherence_config(uint32_t usart_periph, uint32_t bcm);
Function descriptions	configure break frame coherence mode
Precondition	-
The called functions	-

Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Input parameter{in}	
bcm	
USART_BCM_NONE	No parity error is detected
USART_BCM_EN	Parity error is detected
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 break frame coherence mode */

uart_break_frame_coherence_config(USART0, USART_BCM_NONE);
```

uart_parity_check_coherence_config

The description of `uart_parity_check_coherence_config` is shown as below:

Table 3-938. Function `uart_parity_check_coherence_config`

Function name	uart_parity_check_coherence_config
Function prototype	void usart_parity_check_coherence_config(uint32_t usart_periph, uint32_t pcm);
Function descriptions	configure parity check coherence mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
pcm	
USART_PCM_NONE	not check parity
USART_PCM_EN	check the parity
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 parity check coherence mode */
```

```
uart_word_length_setUSART0, USART_PCM_NONE);
```

uart_hardware_flow_coherence_config

The description of `uart_hardware_flow_coherence_config` is shown as below:

Table 3-939. Function `uart_hardware_flow_coherence_config`

Function name	uart_hardware_flow_coherence_config
Function prototype	void usart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);
Function descriptions	configure hardware flow control coherence mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Input parameter{in}	
hcm	
USART_HCM_NONE	nRTS signal equals to the rxne status register
USART_HCM_EN	nRTS signal is set when the last data bit has been sampled
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure hardware flow control coherence mode */
uart_hardware_flow_coherence_configUSART0, USART_HCM_NONE);
```

uart_dma_receive_config

The description of `uart_dma_receive_config` is shown as below:

Table 3-940. Function `uart_dma_receive_config`

Function name	uart_dma_receive_config
Function prototype	void usart_dma_receive_config(uint32_t usart_periph, uint32_t dmacmd);
Function descriptions	configure USART DMA reception
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	

dmacmd	enable or disable DMA for reception
<i>USART_RECEIVE_DM_A_ENABLE</i>	DMA enable for reception
<i>USART_RECEIVE_DM_A_DISABLE</i>	DMA disable for reception
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 DMA enable for reception */

uart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

uart_dma_transmit_config

The description of `uart_dma_transmit_config` is shown as below:

Table 3-941. Function `uart_dma_transmit_config`

Function name	<code>uart_dma_transmit_config</code>
Function prototype	<code>void usart_dma_transmit_config(uint32_t usart_periph, uint32_t dmacmd);</code>
Function descriptions	configure USART DMA transmission
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
dmacmd	enable or disable DMA for transmission
<i>USART_TRANSMIT_DMA_ENABLE</i>	DMA enable for transmission
<i>USART_TRANSMIT_DMA_DISABLE</i>	DMA disable for transmission
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 DMA enable for transmission */

uart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

uart_flag_get

The description of `uart_flag_get` is shown as below:

Table 3-942. Function `uart_flag_get`

Function name	<code>uart_flag_get</code>
Function prototype	<code>FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);</code>
Function descriptions	get flag in STAT0/STAT1/CHC register
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
flag	USART flags, refer to <code>usart_flag_enum</code>
<i>USART_FLAG_CTS</i>	CTS change flag
<i>USART_FLAG_LBD</i>	LIN break detected flag
<i>USART_FLAG_TBE</i>	transmit data buffer empty flag
<i>USART_FLAG_TC</i>	transmission complete flag
<i>USART_FLAG_RBNE</i>	read data buffer not empty flag
<i>USART_FLAG_IDLE</i>	IDLE frame detected flag
<i>USART_FLAG_ORER</i> <i>R</i>	overrun error flag
<i>USART_FLAG_NERR</i>	noise error flag
<i>USART_FLAG_FERR</i>	frame error flag
<i>USART_FLAG_PERR</i>	parity error flag
<i>USART_FLAG_BSY</i>	busy flag
<i>USART_FLAG_EB</i>	end of block flag
<i>USART_FLAG_RT</i>	receiver timeout flag
<i>USART_FLAG_EPERR</i>	early parity error flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get flag USART0 state */

FlagStatus status;

status = usart_flag_get(USART0,USART_FLAG_TBE);
```

uart_flag_clear

The description of `uart_flag_clear` is shown as below:

Table 3-943. Function `uart_flag_clear`

Function name	uart_flag_clear
Function prototype	<code>void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);</code>
Function descriptions	clear flag in STAT0/STAT1 register
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	x=0,1,2,5
<code>UARTx</code>	x=3,4,6,7
Input parameter{in}	
<code>flag</code>	USART flags, refer to <code>usart_flag_enum</code>
<code>USART_FLAG_CTS</code>	CTS change flag
<code>USART_FLAG_LBD</code>	LIN break detected flag
<code>USART_FLAG_TC</code>	transmission complete
<code>USART_FLAG_RBNE</code>	read data buffer not empty
<code>USART_FLAG_EB</code>	end of block flag
<code>USART_FLAG_RT</code>	receiver timeout flag
<code>USART_FLAG_EPERR</code>	early parity error flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear USART0 flag */
uart_flag_clear(USART0,USART_FLAG_TC);
```

uart_interrupt_enable

The description of `uart_interrupt_enable` is shown as below:

Table 3-944. Function `uart_interrupt_enable`

Function name	uart_interrupt_enable
Function prototype	<code>void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);</code>
Function descriptions	enable USART interrupt
Precondition	-
The called functions	-
Input parameter{in}	

usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
interrupt	USART interrupt
<i>USART_INT_PERR</i>	parity error interrupt
<i>USART_INT_TBE</i>	transmitter buffer empty interrupt
<i>USART_INT_TC</i>	transmission complete interrupt
<i>USART_INT_RBNE</i>	read data buffer not empty interrupt and overrun error interrupt
<i>USART_INT_IDLE</i>	IDLE line detected interrupt
<i>USART_INT_LBD</i>	LIN break detected interrupt
<i>USART_INT_ERR</i>	error interrupt
<i>USART_INT_CTS</i>	CTS interrupt
<i>USART_INT_RT</i>	receive timeout event interrupt
<i>USART_INT_EB</i>	end of block event interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 TBE interrupt */
uart_interrupt_enable(USART0, USART_INT_TBE);
```

uart_interrupt_disable

The description of `uart_interrupt_disable` is shown as below:

Table 3-945. Function `uart_interrupt_disable`

Function name	<code>uart_interrupt_disable</code>
Function prototype	<code>void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);</code>
Function descriptions	disable USART interrupt
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
interrupt	USART interrupt flag
<i>USART_INT_PERR</i>	parity error interrupt
<i>USART_INT_TBE</i>	transmitter buffer empty interrupt

USART_INT_TC	transmission complete interrupt
USART_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt
USART_INT_IDLE	IDLE line detected interrupt
USART_INT_LBD	LIN break detected interrupt
USART_INT_ERR	error interrupt
USART_INT_CTS	CTS interrupt
USART_INT_RT	receive timeout event interrupt
USART_INT_EB	end of block event interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 TBE interrupt */

uart_interrupt_disable(USART0, USART_INT_TBE);
```

uart_interrupt_flag_get

The description of `uart_interrupt_flag_get` is shown as below:

Table 3-946. Function `uart_interrupt_flag_get`

Function name	<code>uart_interrupt_flag_get</code>
Function prototype	<code>FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);</code>
Function descriptions	get USART interrupt and flag status
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
int_flag	USART interrupt flag, refer to <code>usart_interrupt_flag_enum</code>
USART_INT_FLAG_PE RR	parity error interrupt and flag
USART_INT_FLAG_TB E	transmitter buffer empty interrupt and flag
USART_INT_FLAG_TC	transmission complete interrupt and flag
USART_INT_FLAG_RB NE	read data buffer not empty interrupt and flag
USART_INT_FLAG_RB NE_ORERR	read data buffer not empty interrupt and overrun error flag

<i>USART_INT_FLAG_IDLE</i>	IDLE line detected interrupt and flag
<i>USART_INT_FLAG_LBD</i>	LIN break detected interrupt and flag
<i>USART_INT_FLAG_CTS</i>	CTS interrupt and flag
<i>USART_INT_FLAG_ERORR</i>	error interrupt and overrun error
<i>USART_INT_FLAG_ERNERR</i>	error interrupt and noise error flag
<i>USART_INT_FLAG_ERFERR</i>	error interrupt and frame error flag
<i>USART_INT_FLAG_EB</i>	end of block event interrupt flag
<i>USART_INT_FLAG_RT</i>	receive timeout event interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the USART0 interrupt flag status */

FlagStatus status;

status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
```

usart_interrupt_flag_clear

The description of usart_interrupt_flag_clear is shown as below:

Table 3-947. Function usart_interrupt_flag_clear

Function name	usart_interrupt_flag_clear
Function prototype	void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
Function descriptions	clear USART interrupt flag in STAT0/STAT1 register
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
int_flag	USART interrupt flag, refer to usart_interrupt_flag_enum
<i>USART_INT_FLAG_CTS</i>	CTS change flag

S	
USART_INT_FLAG_LB D	LIN break detected flag
USART_INT_FLAG_TC	transmission complete
USART_INT_FLAG_RB NE	read data buffer not empty
USART_INT_FLAG_EB	end of block event interrupt flag
USART_INT_FLAG_RT	receive timeout event interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the USART0 interrupt enable bit status */
uart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

3.30. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.30.1](#), the FWDGT firmware functions are introduced in chapter [3.30.2](#).

3.30.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

Table 3-948. WWDGT Registers

Registers	Descriptions
WWDGT_CTL	Control register
WWDGT_CFG	Configuration register
WWDGT_STAT	Status register

3.30.2. Descriptions of Peripheral functions

WWDGT firmware functions are listed in the table shown as below:

Table 3-949. WWDGT firmware function

Function name	Function description
wwdgt_deinit	reset the window watchdog timer configuration

Function name	Function description
wwdgt_enable	start the window watchdog timer counter
wwdgt_counter_update	configure the window watchdog timer counter value
wwdgt_config	configure counter value, window value, and prescaler divider value
wwdgt_interrupt_enable	enable early wakeup interrupt of WWDGT
wwdgt_flag_get	check early wakeup interrupt state of WWDGT
wwdgt_flag_clear	clear early wakeup interrupt state of WWDGT

wwdgt_deinit

The description of wwdgt_deinit is shown as below:

Table 3-950. Function wwdgt_deinit

Function name	wwdgt_deinit
Function prototype	void wwdgt_deinit(void);
Function descriptions	reset the window watchdog timer configuration
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the window watchdog timer configuration */
wwdgt_deinit();
```

wwdgt_enable

The description of wwdgt_enable is shown as below:

Table 3-951. Function wwdgt_enable

Function name	wwdgt_enable
Function prototype	void wwdgt_enable (void);
Function descriptions	start the window watchdog timer counter
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* start the window watchdog timer counter */
wwdgt_enable ( );
```

wwdgt_counter_update

The description of wwdgt_counter_update is shown as below:

Table 3-952. Function wwdgt_counter_update

Function name	wwdgt_counter_update
Function prototype	void wwdgt_counter_update(uint16_t counter_value);
Function descriptions	configure the window watchdog timer counter value
Precondition	-
The called functions	-
Input parameter{in}	
counter_value	0x00 - 0x7F
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* update WWDGT counter to 0x7F */
wwdgt_counter_update(127);
```

wwdgt_config

The description of wwdgt_config is shown as below:

Table 3-953. Function wwdgt_config

Function name	wwdgt_config
Function prototype	void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);
Function descriptions	configure counter value, window value, and prescaler divider value
Precondition	-
The called functions	-
Input parameter{in}	
counter	0x00 - 0x7F
Input parameter{in}	
window	0x00 - 0x7F
Input parameter{in}	
prescaler	wwdgt prescaler value
<i>WWDGT_CFG_PSC_D IV1</i>	the time base of window watchdog counter = (PCLK1/4096)/1
<i>WWDGT_CFG_PSC_D IV2</i>	the time base of window watchdog counter = (PCLK1/4096)/2
<i>WWDGT_CFG_PSC_D IV4</i>	the time base of window watchdog counter = (PCLK1/4096)/4
<i>WWDGT_CFG_PSC_D IV8</i>	the time base of window watchdog counter = (PCLK1/4096)/8
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* confiure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to
8 */

wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

wwdgt_interrupt_enable

The description of wwdgt_interrupt_enable is shown as below:

Table 3-954. Function wwdgt_interrupt_enable

Function name	wwdgt_interrupt_enable
Function prototype	void wwdgt_interrupt_enable(void);
Function descriptions	enable early wakeup interrupt of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable early wakeup interrupt of WWDGT */

wwdgt_interrupt_enable ( );
```

wwdgt_flag_get

The description of wwdgt_flag_get is shown as below:

Table 3-955. Function wwdgt_flag_get

Function name	wwdgt_flag_get
Function prototype	FlagStatus wwdgt_flag_get(void);
Function descriptions	check early wakeup interrupt state of WWDGT

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* test if the counter value update has reached the 0x40 */

FlagStatus status;

status = wwdgt_flag_get ( );

if(status == RESET)

{
    ...

}

}else

{
    ...

}

}
```

wwdgt_flag_clear

The description of wwdgt_flag_clear is shown as below:

Table 3-956. Function wwdgt_flag_clear

Function name	wwdgt_flag_clear
Function prototype	void wwdgt_flag_clear(void);
Function descriptions	clear early wakeup interrupt state of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	

-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear early wakeup interrupt state of WWDGT */  
wwdgt_flag_clear( );
```

4. Revision history

Table 4-1. Revision history

Revison No.	Description	Date
1.0	Initial Release	Dec.28, 2018
1.1	Format update	Jul.7, 2022
1.2	<ol style="list-style-type: none">Modified the description of chapter <u>EXTI</u>Modified the description of chapter <u>DBG</u>Modified the description of chapter <u>SPI</u>Modified the description of chapter <u>TIMER</u>	Dec.28,2022

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company under the intellectual property laws and treaties of the People's Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it's suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it's suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.