

GigaDevice Semiconductor Inc.

GD32F20x
ARM[®] Cortex[®]-M3 32-bit MCU

Firmware Library
User Guide

Revision 1.6

(Feb. 2026)

Table of Contents

Table of Contents	2
List of Figures	5
List of Tables	6
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.....	31
2.1.3. Template Folder	31
2.1.4. Utilities Folder	35
2.2. File descriptions of Firmware Library	35
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. BKP.....	65
3.3.1. Descriptions of Peripheral registers.....	65
3.3.2. Descriptions of Peripheral functions	65
3.4. CAN	79
3.4.1. Descriptions of Peripheral registers.....	79
3.4.2. Descriptions of Peripheral functions	80
3.5. CAU	103
3.5.1. Descriptions of Peripheral registers.....	103
3.5.2. Descriptions of Peripheral functions	103
3.6. CRC	125
3.6.1. Descriptions of Peripheral registers.....	125
3.6.2. Descriptions of Peripheral functions	125
3.7. DAC	130
3.7.1. Peripheral register description	130

3.7.2.	Descriptions of Peripheral functions	130
3.8.	DBG	144
3.8.1.	Descriptions of Peripheral registers	144
3.8.2.	Descriptions of Peripheral functions	145
3.9.	DCI	150
3.9.1.	Descriptions of Peripheral registers	150
3.9.2.	Descriptions of Peripheral functions	151
3.10.	DMA.....	164
3.10.1.	Descriptions of Peripheral registers	164
3.10.2.	Descriptions of Peripheral functions	164
3.11.	ENET	185
3.11.1.	Descriptions of Peripheral registers	186
3.11.2.	Descriptions of Peripheral functions	188
3.12.	EXMC	267
3.12.1.	Descriptions of Peripheral registers	267
3.12.2.	Descriptions of Peripheral functions	268
3.13.	EXTI.....	302
3.13.1.	Descriptions of Peripheral registers	303
3.13.2.	Descriptions of Peripheral functions	303
3.14.	FMC	310
3.14.1.	Descriptions of Peripheral registers	310
3.14.2.	Descriptions of Peripheral functions	311
3.15.	FWDGT.....	329
3.15.1.	Descriptions of Peripheral registers	329
3.15.2.	Descriptions of Peripheral functions	330
3.16.	GPIO	334
3.16.1.	Descriptions of Peripheral registers	334
3.16.2.	Descriptions of Peripheral functions	335
3.17.	HAU	354
3.17.1.	Descriptions of Peripheral registers	355
3.17.2.	Descriptions of Peripheral functions	355
3.18.	I2C	372
3.18.1.	Descriptions of Peripheral registers	372
3.18.2.	Descriptions of Peripheral functions	372
3.19.	MISC.....	393
3.19.1.	Descriptions of Peripheral registers	393
3.19.2.	Descriptions of Peripheral functions	395
3.20.	PMU.....	401
3.20.1.	Descriptions of Peripheral registers	401

3.20.2.	Descriptions of Peripheral functions	402
3.21.	RCU	408
3.21.1.	Descriptions of Peripheral registers	409
3.21.2.	Descriptions of Peripheral functions	409
3.22.	RTC	446
3.22.1.	Descriptions of Peripheral registers	446
3.22.2.	Descriptions of Peripheral functions	446
3.23.	SDIO	454
3.23.1.	Descriptions of Peripheral registers	454
3.23.2.	Descriptions of Peripheral functions	455
3.24.	SPI	485
3.24.1.	Descriptions of Peripheral registers	485
3.24.2.	Descriptions of Peripheral functions	486
3.25.	TIMER	509
3.25.1.	Descriptions of Peripheral registers	509
3.25.2.	Descriptions of Peripheral functions	510
3.26.	TLI	563
3.26.1.	Descriptions of Peripheral registers	564
3.26.2.	Descriptions of Peripheral functions	564
3.27.	TRNG	583
3.27.1.	Descriptions of Peripheral registers	583
3.27.2.	Descriptions of Peripheral functions	583
3.28.	USART	588
3.28.1.	Descriptions of Peripheral registers	588
3.28.2.	Descriptions of Peripheral functions	589
3.29.	WWDGT	621
3.29.1.	Descriptions of Peripheral registers	621
3.29.2.	Descriptions of Peripheral functions	622
4.	Revision history	627

List of Figures

Figure 2-1. File structure of firmware library of GD32F20x	30
Figure 2-2. Select peripheral example files	32
Figure 2-3. Copy the peripheral example files	33
Figure 2-4. Open the project file	34
Figure 2-5. Configure project files	34
Figure 2-6. Compile-debug-download	35

List of Tables

Table 1-1. Peripherals	27
Table 2-1. Function descriptions of Firmware Library	35
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_mode_config	40
Table 3-6. Function adc_special_function_config	41
Table 3-7. Function adc_data_alignment_config	41
Table 3-8. Function adc_enable	42
Table 3-9. Function adc_disable	43
Table 3-10. Function adc_calibration_enable	43
Table 3-11. Function adc_tempsensor_vrefint_enable	44
Table 3-12. Function adc_tempsensor_vrefint_disable	44
Table 3-13. Function adc_dma_mode_enable	45
Table 3-14. Function adc_dma_mode_disable	45
Table 3-15. Function adc_discontinuous_mode_config	46
Table 3-16. Function adc_channel_length_config	46
Table 3-17. Function adc_regular_channel_config	47
Table 3-18. Function adc_inserted_channel_config	48
Table 3-19. Function adc_inserted_channel_offset_config	49
Table 3-20. Function adc_external_trigger_source_config	50
Table 3-21. Function adc_external_trigger_config	52
Table 3-22. Function adc_software_trigger_enable	53
Table 3-23. Function adc_regular_data_read	54
Table 3-24. Function adc_inserted_data_read	54
Table 3-25. Function adc_sync_mode_convert_value_read	55
Table 3-26. Function adc_watchdog_single_channel_enable	55
Table 3-27. Function adc_watchdog_group_channel_enable	56
Table 3-28. Function adc_watchdog_disable	57
Table 3-29. Function adc_watchdog_threshold_config	57
Table 3-30. Function adc_resolution_config	58
Table 3-31. Function adc_oversample_mode_config	58
Table 3-32. Function adc_oversample_mode_enable	60
Table 3-33. Function adc_oversample_mode_disable	61
Table 3-34. Function adc_flag_get	61
Table 3-35. Function adc_flag_clear	62
Table 3-36. Function adc_interrupt_enable	62
Table 3-37. Function adc_interrupt_disable	63
Table 3-38. Function adc_interrupt_flag_get	64

Table 3-39. Function <code>adc_interrupt_flag_clear</code>	64
Table 3-40. BKP Registers	65
Table 3-41. BKP firmware function.....	65
Table 3-42. Enum <code>bkp_data_register_enum</code>	66
Table 3-43. Enum <code>bkp_tamper_enum</code>	67
Table 3-44. Function <code>bkp_deinit</code>	67
Table 3-45. Function <code>bkp_data_write</code>	68
Table 3-46. Function <code>bkp_data_read</code>	68
Table 3-47. Function <code>bkp_rtc_calibration_output_enable</code>	69
Table 3-48. Function <code>bkp_rtc_calibration_output_disable</code>	70
Table 3-49. Function <code>bkp_rtc_signal_output_enable</code>	70
Table 3-50. Function <code>bkp_rtc_signal_output_disable</code>	70
Table 3-51. Function <code>bkp_rtc_output_select</code>	71
Table 3-52. Function <code>bkp_rtc_clock_output_select</code>	72
Table 3-53. Function <code>bkp_rtc_clock_calibration_direction</code>	72
Table 3-54. Function <code>bkp_rtc_calibration_value_set</code>	73
Table 3-55. Function <code>bkp_tamper_detection_enable</code>	73
Table 3-56. Function <code>bkp_tamper_detection_disable</code>	74
Table 3-57. Function <code>bkp_tamper_active_level_set</code>	74
Table 3-58. Function <code>bkp_waveform_detect_config</code>	75
Table 3-59. Function <code>bkp_flag_get</code>	76
Table 3-60. Function <code>bkp_flag_clear</code>	76
Table 3-61. Function <code>bkp_tamper_interrupt_enable</code>	77
Table 3-62. Function <code>bkp_tamper_interrupt_disable</code>	77
Table 3-63. Function <code>bkp_interrupt_flag_get</code>	78
Table 3-64. Function <code>bkp_interrupt_flag_clear</code>	78
Table 3-65. CAN Registers	79
Table 3-66. CAN firmware function	80
Table 3-67. Structure <code>can_parameter_struct</code>	81
Table 3-68. Structure <code>can_transmit_message_struct</code>	81
Table 3-69. Structure <code>can_receive_message_struct</code>	81
Table 3-70. Structure <code>can_filter_parameter_struct</code>	82
Table 3-71. Enum <code>can_flag_enum</code>	82
Table 3-72. Enum <code>can_interrupt_flag_enum</code>	83
Table 3-73. Enum <code>can_error_enum</code>	83
Table 3-74. Enum <code>can_transmit_state_enum</code>	84
Table 3-75. Enum <code>can_struct_type_enum</code>	84
Table 3-76. Function <code>can_deinit</code>	84
Table 3-77. Function <code>can_struct_para_init</code>	85
Table 3-78. Function <code>can_init</code>	85
Table 3-79. Function <code>can_filter_init</code>	86
Table 3-80. Function <code>can1_filter_start_bank</code>	87
Table 3-81. Function <code>can_debug_freeze_enable</code>	88
Table 3-82. Function <code>can_debug_freeze_disable</code>	88

Table 3-83. Function can_time_trigger_mode_enable.....	89
Table 3-84. Function can_time_trigger_mode_disable.....	89
Table 3-85. Function can_message_transmit.....	90
Table 3-86. Function can_transmit_states.....	91
Table 3-87. Function can_transmission_stop.....	91
Table 3-88. Function can_message_receive.....	92
Table 3-89. Function can_fifo_release.....	93
Table 3-90. Function can_receive_message_length_get.....	93
Table 3-91. Function can_working_mode_set.....	94
Table 3-92. Function can_wakeup.....	94
Table 3-93. Function can_error_get.....	95
Table 3-94. Function can_receive_error_number_get.....	95
Table 3-95. Function can_transmit_error_number_get.....	96
Table 3-96. Function can_flag_get.....	97
Table 3-97. Function can_flag_clear.....	98
Table 3-98. Function can_interrupt_enable.....	99
Table 3-99. Function can_interrupt_disable.....	100
Table 3-100. Function can_interrupt_flag_get.....	101
Table 3-101. Function can_interrupt_flag_clear.....	102
Table 3-102. CAU Registers.....	103
Table 3-103. CAU firmware function.....	104
Table 3-104. Structure cau_key_parameter_struct.....	104
Table 3-105. Structure cau_iv_parameter_struct.....	105
Table 3-106. Enum cau_text_struct.....	105
Table 3-107. Function cau_deinit.....	105
Table 3-108. Function cau_enable.....	106
Table 3-109. Function cau_disable.....	106
Table 3-110. Function cau_dma_enable.....	106
Table 3-111. Function cau_dma_disable.....	107
Table 3-112. Function cau_init.....	108
Table 3-113. Function cau_aes_keysize_config.....	109
Table 3-114. Function cau_key_init.....	109
Table 3-115. Function cau_key_parameter_init.....	110
Table 3-116. Function cau_iv_init.....	110
Table 3-117. Function cau_iv_parameter_init.....	111
Table 3-118. Function cau_fifo_flush.....	112
Table 3-119. Function cau_enable_state_get.....	112
Table 3-120. Function cau_data_write.....	113
Table 3-121. Function cau_data_read.....	113
Table 3-122. Function cau_aes_ecb.....	114
Table 3-123. Function cau_aes_cbc.....	115
Table 3-124. Function cau_aes_ctr.....	116
Table 3-125. Function cau_tdes_ecb.....	118
Table 3-126. Function cau_tdes_cbc.....	119

Table 3-127. Function cau_des_ecb.....	120
Table 3-128. Function cau_des_cbc.....	121
Table 3-129. Function cau_flag_get	123
Table 3-130. Function cau_interrupt_enable	123
Table 3-131. Function cau_interrupt_disable	124
Table 3-132. Function cau_interrupt_flag_get.....	124
Table 3-133. CRC Registers	125
Table 3-134. CRC firmware function	125
Table 3-135. Function crc_deinit	126
Table 3-136. Function crc_data_register_reset.....	126
Table 3-137. Function crc_data_register_read.....	127
Table 3-138. Function crc_free_data_register_read.....	127
Table 3-139. Function crc_free_data_register_write	128
Table 3-140. Function crc_single_data_calculate	128
Table 3-141. Function crc_block_data_calculate	129
Table 3-142. DAC Registers	130
Table 3-143. DAC firmware functions	130
Table 3-144. Function dac_deinit.....	131
Table 3-145. Function dac_enable.....	131
Table 3-146. Function dac_disable.....	132
Table 3-147. Function dac_dma_enable.....	133
Table 3-148. Function dac_dma_disable	133
Table 3-149. Function dac_output_buffer_enable	134
Table 3-150. Function dac_output_buffer_disable	134
Table 3-151. Function dac_output_value_get	135
Table 3-152. Function dac_data_set	135
Table 3-153. Function dac_trigger_enable.....	136
Table 3-154. Function dac_trigger_disable.....	137
Table 3-155. Function dac_trigger_source_config.....	137
Table 3-156. Function dac_software_trigger_enable	138
Table 3-157. Function dac_wave_mode_config.....	139
Table 3-158. Function dac_lfsr_noise_config	140
Table 3-159. Function dac_triangle_noise_config.....	140
Table 3-160. Function dac_concurrent_enable.....	141
Table 3-161. Function dac_concurrent_disable	142
Table 3-162. Function dac_concurrent_software_trigger_enable	142
Table 3-163. Function dac_concurrent_output_buffer_enable	143
Table 3-164. Function dac_concurrent_output_buffer_disable	143
Table 3-165. Function dac_concurrent_data_set	144
Table 3-166. DBG Registers	145
Table 3-167. DBG firmware function	145
Table 3-168. Enum dbg_periph_enum	145
Table 3-169. Function dbg_id_get	146
Table 3-170. Function dbg_low_power_enable.....	146

Table 3-171. Function <code>dbg_low_power_disable</code>	147
Table 3-172. Function <code>dbg_periph_enable</code>	147
Table 3-173. Function <code>dbg_periph_disable</code>	148
Table 3-174. Function <code>dbg_trace_pin_enable</code>	149
Table 3-175. Function <code>dbg_trace_pin_disable</code>	149
Table 3-176. Function <code>dbg_trace_pin_mode_set</code>	150
Table 3-177. DCI Registers.....	150
Table 3-178. DCI firmware function	151
Table 3-179. Structure <code>dc_i_parameter_struct</code>	152
Table 3-180. Function <code>dc_i_deinit</code>	152
Table 3-181. Function <code>dc_i_init</code>	153
Table 3-182. Function <code>dc_i_enable</code>	153
Table 3-183. Function <code>dc_i_disable</code>	154
Table 3-184. Function <code>dc_i_capture_enable</code>	154
Table 3-185. Function <code>dc_i_capture_disable</code>	155
Table 3-186. Function <code>dc_i_jpeg_enable</code>	155
Table 3-187. Function <code>dc_i_jpeg_disable</code>	156
Table 3-188. Function <code>dc_i_crop_window_enable</code>	156
Table 3-189. Function <code>dc_i_crop_window_disable</code>	157
Table 3-190. Function <code>dc_i_crop_window_config</code>	157
Table 3-191. Function <code>dc_i_embedded_sync_enable</code>	158
Table 3-192. Function <code>dc_i_embedded_sync_disable</code>	158
Table 3-193. Function <code>dc_i_sync_codes_config</code>	159
Table 3-194. Function <code>dc_i_sync_codes_unmask_config</code>	159
Table 3-195. Function <code>dc_i_data_read</code>	160
Table 3-196. Function <code>dc_i_flag_get</code>	160
Table 3-197. Function <code>dc_i_interrupt_enable</code>	161
Table 3-198. Function <code>dc_i_interrupt_disable</code>	162
Table 3-199. Function <code>dc_i_interrupt_flag_get</code>	162
Table 3-200. Function <code>dc_i_interrupt_flag_clear</code>	163
Table 3-201. DMA Registers.....	164
Table 3-202. DMA firmware function	164
Table 3-203. Structure <code>dma_parameter_struct</code>	165
Table 3-204. Function <code>dma_deinit</code>	166
Table 3-205. Function <code>dma_struct_para_init</code>	166
Table 3-206. Function <code>dma_init</code>	167
Table 3-207. Function <code>dma_circulation_enable</code>	168
Table 3-208. Function <code>dma_circulation_disable</code>	168
Table 3-209. Function <code>dma_memory_to_memory_enable</code>	169
Table 3-210. Function <code>dma_memory_to_memory_disable</code>	170
Table 3-211. Function <code>dma_channel_enable</code>	170
Table 3-212. Function <code>dma_channel_disable</code>	171
Table 3-213. Function <code>dma_periph_address_config</code>	171
Table 3-214. Function <code>dma_memory_address_config</code>	172

Table 3-215. Function dma_transfer_number_config	173
Table 3-216. Function dma_transfer_number_get.....	174
Table 3-217. Function dma_priority_config	174
Table 3-218. Function dma_memory_width_config	175
Table 3-219. Function dma_periph_width_config	176
Table 3-220. Function dma_memory_increase_enable	177
Table 3-221. Function dma_memory_increase_disable	177
Table 3-222. Function dma_periph_increase_enable	178
Table 3-223. Function dma_periph_increase_disable	179
Table 3-224. Function dma_transfer_direction_config.....	179
Table 3-225. Function dma_1_channel_5_fulldata_transfer_enable	180
Table 3-226. Function dma_1_channel_5_fulldata_transfer_disable	180
Table 3-227. Function dma_flag_get	181
Table 3-228. Function dma_flag_clear.....	182
Table 3-229. Function dma_interrupt_enable.....	182
Table 3-230. Function dma_interrupt_disable.....	183
Table 3-231. Function dma_interrupt_flag_get	184
Table 3-232. Function dma_interrupt_flag_clear	185
Table 3-233. ENET Registers	186
Table 3-234. ENET firmware function	188
Table 3-235. Structure enet_descriptors_struct	191
Table 3-236. Structure enet_ptp_systime_struct.....	191
Table 3-237. Enum enet_flag_enum	191
Table 3-238. Enum enet_flag_clear_enum	193
Table 3-239. Enum enet_int_enum	194
Table 3-240. Enum enet_int_flag_enum	195
Table 3-241. Enum enet_int_flag_clear_enum	197
Table 3-242. Enum enet_desc_reg_enum	197
Table 3-243. Enum enet_msc_counter_enum	198
Table 3-244. Enum enet_option_enum	198
Table 3-245. Enum enet_mediamode_enum.....	199
Table 3-246. Enum enet_chksumconf_enum	199
Table 3-247. Enum enet_frmrecept_enum	199
Table 3-248. Enum enet_registers_type_enum.....	200
Table 3-249. Enum enet_dmadirection_enum	200
Table 3-250. Enum enet_phydirection_enum	200
Table 3-251. Enum enet_regdirection_enum.....	200
Table 3-252. Enum enet_macaddress_enum.....	200
Table 3-253. Enum enet_descstate_enum	201
Table 3-254. Function enet_deinit	201
Table 3-255. Function enet_initpara_config	202
Table 3-256. Function enet_init.....	205
Table 3-257. Function enet_software_reset.....	206
Table 3-258. Function enet_rxframe_size_get.....	207

Table 3-259. Function enet_descriptors_chain_init	207
Table 3-260. Function enet_descriptors_ring_init.....	208
Table 3-261. Function enet_frame_receive	209
Table 3-262. Function enet_frame_transmit	209
Table 3-263. Function enet_transmit_checksum_config	210
Table 3-264. Function enet_enable	211
Table 3-265. Function enet_disable	211
Table 3-266. Function enet_mac_address_set.....	211
Table 3-267. Function enet_mac_address_get	213
Table 3-268. Function enet_flag_get.....	213
Table 3-269. Function enet_flag_clear	215
Table 3-270. Function enet_interrupt_enable.....	217
Table 3-271. Function enet_interrupt_disable.....	218
Table 3-272. Function enet_interrupt_flag_get	219
Table 3-273. Function enet_interrupt_flag_clear	221
Table 3-274. Function enet_tx_enable	222
Table 3-275. Function enet_tx_disable	223
Table 3-276. Function enet_rx_enable	223
Table 3-277. Function enet_rx_disable	224
Table 3-278. Function enet_registers_get.....	224
Table 3-279. Function enet_address_filter_enable.....	225
Table 3-280. Function enet_address_filter_disable	226
Table 3-281. Function enet_address_filter_config	227
Table 3-282. Function enet_phy_config	228
Table 3-283. Function enet_phy_write_read.....	228
Table 3-284. Function enet_phyloopback_enable	229
Table 3-285. Function enet_phyloopback_disable	230
Table 3-286. Function enet_forward_feature_enable	230
Table 3-287. Function enet_forward_feature_disable	231
Table 3-288. Function enet_fliter_feature_enable	231
Table 3-289. Function enet_fliter_feature_disable	232
Table 3-290. Function enet_pauseframe_generate	233
Table 3-291. Function enet_pauseframe_detect_config	234
Table 3-292. Function enet_pauseframe_config	234
Table 3-293. Function enet_flowcontrol_threshold_config	235
Table 3-294. Function enet_flowcontrol_feature_enable	236
Table 3-295. Function enet_flowcontrol_feature_disable	237
Table 3-296. Function enet_dmaprocess_state_get	238
Table 3-297. Function enet_dmaprocess_resume.....	239
Table 3-298. Function enet_rxprocess_check_recovery.....	239
Table 3-299. Function enet_txfifo_flush	240
Table 3-300. Function enet_current_desc_address_get	240
Table 3-301. Function enet_desc_information_get	241
Table 3-302. Function enet_missed_frame_counter_get	242

Table 3-303. Function enet_desc_flag_get	243
Table 3-304. Function enet_desc_flag_set	244
Table 3-305. Function enet_desc_flag_clear	245
Table 3-306. Function enet_desc_receive_complete_bit_enable	246
Table 3-307. Function enet_desc_receive_complete_bit_disable	247
Table 3-308. Function enet_rxframe_drop	247
Table 3-309. Function enet_dma_feature_enable	248
Table 3-310. Function enet_dma_feature_disable	249
Table 3-311. Function enet_ptp_normal_descriptors_chain_init	249
Table 3-312. Function enet_ptp_normal_descriptors_ring_init	250
Table 3-313. Function enet_ptpframe_receive_normal_mode	251
Table 3-314. Function enet_ptpframe_transmit_normal_mode	251
Table 3-315. Function enet_wum_filter_register_pointer_reset	252
Table 3-316. Function enet_wum_filter_config	252
Table 3-317. Function enet_wum_feature_enable	253
Table 3-318. Function enet_wum_feature_disable	254
Table 3-319. Function enet_msc_counters_reset	254
Table 3-320. Function enet_msc_feature_enable	255
Table 3-321. Function enet_msc_feature_disable	255
Table 3-322. Function enet_msc_counters_get	256
Table 3-323. Function enet_ptp_subsecond_2_nanosecond	257
Table 3-324. Function enet_ptp_nanosecond_2_subsecond	257
Table 3-325. Function enet_ptp_feature_enable	258
Table 3-326. Function enet_ptp_feature_disable	258
Table 3-327. Function enet_ptp_timestamp_function_config	259
Table 3-328. Function enet_ptp_subsecond_increment_config	260
Table 3-329. Function enet_ptp_timestamp_addend_config	260
Table 3-330. Function enet_ptp_timestamp_update_config	261
Table 3-331. Function enet_ptp_expected_time_config	262
Table 3-332. Function enet_ptp_system_time_get	262
Table 3-333. Function enet_ptp_start	263
Table 3-334. Function enet_ptp_finecorrection_adjfreq	264
Table 3-335. Function enet_ptp_coarsecorrection_systime_update	264
Table 3-336. Function enet_ptp_finecorrection_settime	265
Table 3-337. Function enet_ptp_flag_get	266
Table 3-338. Function enet_initpara_reset	266
Table 3-339. EXMC Registers	267
Table 3-340. EXMC firmware function	268
Table 3-341. Structure exmc_norsram_timing_parameter_struct	269
Table 3-342. Structure exmc_norsram_parameter_struct	269
Table 3-343. Structure exmc_nand_pccard_timing_parameter_struct	270
Table 3-344. Structure exmc_nand_parameter_struct	270
Table 3-345. Structure exmc_pccard_parameter_struct	271
Table 3-346. Structure exmc_sdram_timing_parameter_struct	271

Table 3-347. Structure exmc_sdram_parameter_struct	271
Table 3-348. Structure exmc_sdram_command_parameter_struct	272
Table 3-349. Structure exmc_sqpsram_parameter_struct.....	272
Table 3-350. Function exmc_norsram_deinit	272
Table 3-351. Function exmc_norsram_struct_para_init.....	273
Table 3-352. Function exmc_norsram_init.....	273
Table 3-353. Function exmc_norsram_enable	275
Table 3-354. Function exmc_norsram_disable	275
Table 3-355. Function exmc_nand_deinit	276
Table 3-356. Function exmc_nand_struct_para_init.....	276
Table 3-357. Function exmc_nand_init.....	277
Table 3-358. Function exmc_nand_enable	278
Table 3-359. Function exmc_nand_disable	279
Table 3-360. Function exmc_nand_ecc_config.....	279
Table 3-361. Function exmc_ecc_get	280
Table 3-362. Function exmc_pccard_deinit.....	280
Table 3-363. Function exmc_pccard_struct_para_init.....	281
Table 3-364. Function exmc_pccard_init	281
Table 3-365. Function exmc_pccard_enable	282
Table 3-366. Function exmc_pccard_disable.....	283
Table 3-367. Function exmc_sdram_deinit.....	283
Table 3-368. Function exmc_sdram_struct_para_init.....	284
Table 3-369. Function exmc_sdram_init	284
Table 3-370. Function exmc_sdram_struct_command_para_init.....	285
Table 3-371. Function exmc_sdram_command_config.....	286
Table 3-372. Function exmc_sdram_refresh_count_set	287
Table 3-373. Function exmc_sdram_autorefresh_number_set.....	287
Table 3-374. Function exmc_sdram_write_protection_config	288
Table 3-375. Function exmc_sdram_bankstatus_get	288
Table 3-376. Function exmc_sdram_readsample_config.....	289
Table 3-377. Function exmc_sdram_readsample_enable	290
Table 3-378. Function exmc_sdram_readsample_disable	290
Table 3-379. Function exmc_sqpsram_deinit	291
Table 3-380. Function exmc_sqpsram_struct_para_init.....	291
Table 3-381. Function exmc_sqpsram_init.....	292
Table 3-382. Function exmc_sqpsram_read_command_set.....	292
Table 3-383. Function exmc_sqpsram_write_command_set.....	293
Table 3-384. Function exmc_sqpsram_read_id_command_send	294
Table 3-385. Function exmc_sqpsram_write_cmd_send.....	295
Table 3-386. Function exmc_sqpsram_low_id_get.....	295
Table 3-387. Function exmc_sqpsram_low_id_get.....	296
Table 3-388. Function exmc_sqpsram_send_command_state_get.....	296
Table 3-389. Function exmc_flag_get	297
Table 3-390. Function exmc_flag_clear	298

Table 3-391. Function <code>exmc_interrupt_enable</code>	299
Table 3-392. Function <code>exmc_interrupt_disable</code>	300
Table 3-393. Function <code>exmc_interrupt_flag_get</code>	301
Table 3-394. Function <code>exmc_interrupt_flag_clear</code>	302
Table 3-395. EXTI Registers	303
Table 3-396. EXTI firmware function	303
Table 3-397. Enum <code>exti_line_enum</code>	303
Table 3-398. Enum <code>exti_mode_enum</code>	304
Table 3-399. Enum <code>exti_trig_type_enum</code>	304
Table 3-400. Function <code>exti_deinit</code>	304
Table 3-401. Function <code>exti_init</code>	305
Table 3-402. Function <code>exti_interrupt_enable</code>	305
Table 3-403. Function <code>exti_interrupt_disable</code>	306
Table 3-404. Function <code>exti_event_enable</code>	306
Table 3-405. Function <code>exti_event_disable</code>	307
Table 3-406. Function <code>exti_software_interrupt_enable</code>	307
Table 3-407. Function <code>exti_software_interrupt_disable</code>	308
Table 3-408. Function <code>exti_flag_get</code>	308
Table 3-409. Function <code>exti_flag_clear</code>	309
Table 3-410. Function <code>exti_interrupt_flag_get</code>	309
Table 3-411. Function <code>exti_interrupt_flag_clear</code>	310
Table 3-412. FMC Registers	310
Table 3-413. FMC firmware function	311
Table 3-414. Enum <code>fmc_state_enum</code>	312
Table 3-415. Enum <code>fmc_interrupt_enum</code>	312
Table 3-416. Enum <code>fmc_flag_enum</code>	312
Table 3-417. Enum <code>fmc_interrupt_flag_enum</code>	313
Table 3-418. Function <code>fmc_wscnt_set</code>	313
Table 3-419. Function <code>fmc_unlock</code>	314
Table 3-420. Function <code>fmc_bank0_unlock</code>	314
Table 3-421. Function <code>fmc_bank1_unlock</code>	315
Table 3-422. Function <code>fmc_lock</code>	315
Table 3-423. Function <code>fmc_bank0_lock</code>	316
Table 3-424. Function <code>fmc_bank1_lock</code>	316
Table 3-425. Function <code>fmc_page_erase</code>	317
Table 3-426. Function <code>fmc_mass_erase</code>	317
Table 3-427. Function <code>fmc_bank0_erase</code>	318
Table 3-428. Function <code>fmc_bank1_erase</code>	318
Table 3-429. Function <code>fmc_word_program</code>	319
Table 3-430. Function <code>fmc_halfword_program</code>	319
Table 3-431. Function <code>ob_unlock</code>	320
Table 3-432. Function <code>ob_lock</code>	320
Table 3-433. Function <code>ob_erase</code>	321
Table 3-434. Function <code>ob_write_protection_enable</code>	321

Table 3-435. Function ob_security_protection_config	322
Table 3-436. Function ob_user_write	322
Table 3-437. Function ob_data_program.....	323
Table 3-438. Function ob_user_get	324
Table 3-439. Function ob_data_get	324
Table 3-440. Function ob_write_protection_get.....	325
Table 3-441. Function ob_spc_get.....	325
Table 3-442. Function fmc_flag_get	326
Table 3-443. Function fmc_flag_clear	326
Table 3-444. Function fmc_interrupt_enable	327
Table 3-445. Function fmc_interrupt_disable	328
Table 3-446. Function fmc_interrupt_flag_get	328
Table 3-447. Function fmc_interrupt_flag_clear	329
Table 3-448. FWDGT Registers	329
Table 3-449. FWDGT firmware function	330
Table 3-450. Function fwdgt_write_enable	330
Table 3-451. Function fwdgt_write_disable	330
Table 3-452. Function fwdgt_enable	331
Table 3-453. Function fwdgt_prescaler_value_config	331
Table 3-454. Function fwdgt_reload_value_config.....	332
Table 3-455. Function fwdgt_counter_reload.....	333
Table 3-456. Function fwdgt_config.....	333
Table 3-457. Function fwdgt_flag_get fwdgt_write_disable	334
Table 3-458. GPIO Registers.....	334
Table 3-459. GPIO firmware function	335
Table 3-460. Function gpio_deinit	336
Table 3-461. Function gpio_afio_deinit	336
Table 3-462. Function gpio_init.....	337
Table 3-463. Function gpio_bit_set	338
Table 3-464. Function gpio_bit_reset	338
Table 3-465. Function gpio_bit_write.....	339
Table 3-466. Function gpio_port_write	340
Table 3-467. Function gpio_input_bit_get.....	340
Table 3-468. Function gpio_input_port_get.....	341
Table 3-469. Function gpio_output_bit_get	341
Table 3-470. Function gpio_output_port_get	342
Table 3-471. Function gpio_pin_remap_config.....	342
Table 3-472. Function gpio_pin_remap1_config.....	345
Table 3-473. Function gpio_ethernet_phy_select.....	351
Table 3-474. Function gpio_exti_source_select	352
Table 3-475. Function gpio_event_output_config	352
Table 3-476. Function gpio_event_output_enable	353
Table 3-477. Function gpio_event_output_disable	353
Table 3-478. Function gpio_pin_lock	354

Table 3-479. HAU Registers	355
Table 3-480. HAU firmware function	355
Table 3-481. Structure hau_init_parameter_struct	356
Table 3-482. Structure hau_digest_parameter_struct	356
Table 3-483. Function hau_deinit	356
Table 3-484. Function hau_init	357
Table 3-485. Function hau_init_parameter_init	357
Table 3-486. Function hau_reset	358
Table 3-487. Function hau_last_word_validbits_num_config	359
Table 3-488. Function hau_data_write	359
Table 3-489. Function hau_infifo_words_num_get	359
Table 3-490. Function hau_digest_read	360
Table 3-491. Function hau_digest_calculation_enable	361
Table 3-492. Function hau_multiple_single_dma_config	361
Table 3-493. Function hau_dma_enable	362
Table 3-494. Function hau_dma_disable	362
Table 3-495. Function hau_hash_sha_1	363
Table 3-496. Function hau_hmac_sha_1	363
Table 3-497. Function hau_hash_sha_224	364
Table 3-498. Function hau_hmac_sha_224	365
Table 3-499. Function hau_hash_sha_256	366
Table 3-500. Function hau_hmac_sha_256	366
Table 3-501. Function hau_hash_md5	367
Table 3-502. Function hau_hmac_md5	368
Table 3-503. Function hau_flag_get	368
Table 3-504. Function hau_flag_clear	369
Table 3-505. Function hau_interrupt_enable	370
Table 3-506. Function hau_interrupt_disable	370
Table 3-507. Function hau_interrupt_flag_get	371
Table 3-508. Function hau_interrupt_flag_clear	371
Table 3-509. I2C Registers	372
Table 3-510. I2C firmware function	373
Table 3-511. Function i2c_deinit	373
Table 3-512. Function i2c_clock_config	374
Table 3-513. Function i2c_mode_addr_config	375
Table 3-514. Function i2c_smbus_type_config	375
Table 3-515. Function i2c_ack_config	376
Table 3-516. Function i2c_ackpos_config	377
Table 3-517. Function i2c_master_addressing	377
Table 3-518. Function i2c_dualaddr_enable	378
Table 3-519. Function i2c_dualaddr_disable	378
Table 3-520. Function i2c_enable	379
Table 3-521. Function i2c_disable	379
Table 3-522. Function i2c_start_on_bus	380

Table 3-523. Function i2c_stop_on_bus	380
Table 3-524. Function i2c_data_transmit	381
Table 3-525. Function i2c_data_receive	382
Table 3-526. Function i2c_dma_enable	382
Table 3-527. Function i2c_dma_last_transfer_config	383
Table 3-528. Function i2c_stretch_scl_low_config	383
Table 3-529. Function i2c_slave_response_to_gcall_config	384
Table 3-530. Function i2c_software_reset_config	385
Table 3-531. Function i2c_pec_enable	385
Table 3-532. Function i2c_pec_transfer_enable	386
Table 3-533. Function i2c_pec_value_get	386
Table 3-534. Function i2c_smbus_issue_alert	387
Table 3-535. Function i2c_smbus_arp_enable	388
Table 3-536. Function i2c_flag_get	388
Table 3-537. Function i2c_flag_clear	389
Table 3-538. Function i2c_interrupt_enable	390
Table 3-539. Function i2c_interrupt_disable	391
Table 3-540. Function i2c_interrupt_flag_get	391
Table 3-541. Function i2c_interrupt_flag_clear	392
Table 3-542. NVIC Registers	393
Table 3-543. SysTick Registers	394
Table 3-544. IRQn_Type	395
Table 3-545. MISC firmware function	397
Table 3-546. Function nvic_priority_group_set	397
Table 3-547. Function nvic_irq_enable	398
Table 3-548. Function nvic_irq_disable	398
Table 3-549. Function nvic_vector_table_set	399
Table 3-550. Function system_lowpower_set	399
Table 3-551. Function system_lowpower_reset	400
Table 3-552. Function systick_clksource_set	401
Table 3-553. PMU Registers	402
Table 3-554. PMU firmware function	402
Table 3-555. Function pmu_deinit	402
Table 3-556. Function pmu_lvd_select	403
Table 3-557. Function pmu_lvd_disable	403
Table 3-558. Function pmu_to_sleepmode	404
Table 3-559. Function pmu_to_deepsleepmode	404
Table 3-560. Function pmu_to_standbymode	405
Table 3-561. Function pmu_wakeup_pin_enable	405
Table 3-562. Function pmu_wakeup_pin_disable	406
Table 3-563. Function pmu_backup_write_enable	406
Table 3-564. Function pmu_backup_write_disable	407
Table 3-565. Function pmu_flag_get	407
Table 3-566. Function pmu_flag_clear	408

Table 3-567. RCU Registers	409
Table 3-568. RCU firmware function	409
Table 3-569. Enum rcu_periph_enum	411
Table 3-570. Enum rcu_periph_sleep_enum	411
Table 3-571. Enum rcu_periph_reset_enum	412
Table 3-572. Enum rcu_flag_enum	412
Table 3-573. Enum rcu_int_flag_enum	413
Table 3-574. Enum rcu_int_flag_clear_enum	413
Table 3-575. Enum rcu_int_enum	414
Table 3-576. Enum rcu_osci_type_enum	414
Table 3-577. Enum rcu_clock_freq_enum	415
Table 3-578. Function rcu_deinit	415
Table 3-579. Function rcu_periph_clock_enable	415
Table 3-580. Function rcu_periph_clock_disable	417
Table 3-581. Function rcu_periph_clock_sleep_enable	418
Table 3-582. Function rcu_periph_clock_sleep_disable	418
Table 3-583. Function rcu_periph_reset_enable	419
Table 3-584. Function rcu_periph_reset_disable	420
Table 3-585. Function rcu_bkp_reset_enable	421
Table 3-586. Function rcu_bkp_reset_disable	421
Table 3-587. Function rcu_system_clock_source_config	422
Table 3-588. Function rcu_system_clock_source_get	422
Table 3-589. Function rcu_ahb_clock_config	423
Table 3-590. Function rcu_apb1_clock_config	423
Table 3-591. Function rcu_apb2_clock_config	424
Table 3-592. Function rcu_ckout0_config	425
Table 3-593. Function rcu_ckout1_config	426
Table 3-594. Function rcu_pll_config	427
Table 3-595. Function rcu_predv0_config	427
Table 3-596. Function rcu_predv1_config	428
Table 3-597. Function rcu_pll1_config	429
Table 3-598. Function rcu_pll2_config	429
Table 3-599. Function rcu_adc_clock_config	430
Table 3-600. Function rcu_usbfs_trng_clock_config	430
Table 3-601. Function rcu_rtc_clock_config	431
Table 3-602. Function rcu_i2s1_clock_config	432
Table 3-603. Function rcu_i2s2_clock_config	432
Table 3-604. Function rcu_pllt_config	433
Table 3-605. Function rcu_pllt_vco_config	433
Table 3-606. Function rcu_tli_clock_config	434
Table 3-607. Function rcu_lxtal_drive_capability_config	435
Table 3-608. Function rcu_osci_stab_wait	435
Table 3-609. Function rcu_osci_on	436
Table 3-610. Function rcu_osci_off	437

Table 3-611. Function rcu_osc_bypass_mode_enable.....	437
Table 3-612. Function rcu_osc_bypass_mode_disable	438
Table 3-613. Function rcu_hxtal_clock_monitor_enable	438
Table 3-614. Function rcu_hxtal_clock_monitor_disable	439
Table 3-615. Function rcu_irc8m_adjust_value_set.....	439
Table 3-616. Function rcu_deepsleep_voltage_set.....	440
Table 3-617. Function rcu_clock_freq_get.....	440
Table 3-618. Function rcu_flag_get.....	441
Table 3-619. Function rcu_all_reset_flag_clear	442
Table 3-620. Function rcu_interrupt_enable.....	443
Table 3-621. Function rcu_interrupt_disable.....	443
Table 3-622. Function rcu_interrupt_flag_get	444
Table 3-623. Function rcu_interrupt_flag_clear	445
Table 3-624. RTC Registers	446
Table 3-625. RTC firmware function.....	446
Table 3-626. Function rtc_configuration_mode_enter.....	447
Table 3-627. Function rtc_configuration_mode_exit	447
Table 3-628. Function rtc_lwoff_wait	448
Table 3-629. Function rtc_register_sync_wait	448
Table 3-630. Function rtc_counter_get.....	449
Table 3-631. Function rtc_counter_set.....	449
Table 3-632. Function rtc_prescaler_set	450
Table 3-633. Function rtc_alarm_config.....	450
Table 3-634. Function rtc_divider_get	451
Table 3-635. Function rtc_flag_get.....	451
Table 3-636. Function rtc_flag_clear.....	452
Table 3-637. Function rtc_interrupt_enable.....	453
Table 3-638. Function rtc_interrupt_disable.....	453
Table 3-639. SDIO Registers.....	454
Table 3-640. SDIO firmware function	455
Table 3-641. Function sdio_deinit	456
Table 3-642. Function sdio_clock_config	457
Table 3-643. Function sdio_hardware_clock_enable.....	458
Table 3-644. Function sdio_hardware_clock_disable.....	458
Table 3-645. Function sdio_bus_mode_set	459
Table 3-646. Function sdio_power_state_set.....	459
Table 3-647. Function sdio_power_state_get.....	460
Table 3-648. Function sdio_clock_enable.....	460
Table 3-649. Function sdio_clock_disable.....	461
Table 3-650. Function sdio_command_response_config	461
Table 3-651. Function sdio_wait_type_set.....	462
Table 3-652. Function sdio_csm_enable.....	463
Table 3-653. Function sdio_csm_disable.....	463
Table 3-654. Function sdio_command_index_get.....	464

Table 3-655. Function <code>sdio_response_get</code>	464
Table 3-656. Function <code>sdio_data_config</code>	465
Table 3-657. Function <code>sdio_data_transfer_config</code>	466
Table 3-658. Function <code>sdio_dsm_enable</code>	467
Table 3-659. Function <code>sdio_dsm_disable</code>	467
Table 3-660. Function <code>sdio_data_write</code>	468
Table 3-661. Function <code>sdio_data_read</code>	468
Table 3-662. Function <code>sdio_data_counter_get</code>	469
Table 3-663. Function <code>sdio_data_counter_get</code>	469
Table 3-664. Function <code>sdio_dma_enable</code>	470
Table 3-665. Function <code>sdio_dma_disable</code>	470
Table 3-666. Function <code>sdio_readwait_enable</code>	471
Table 3-667. Function <code>sdio_readwait_disable</code>	471
Table 3-668. Function <code>sdio_stop_readwait_enable</code>	472
Table 3-669. Function <code>sdio_stop_readwait_disable</code>	472
Table 3-670. Function <code>sdio_readwait_type_set</code>	473
Table 3-671. Function <code>sdio_operation_enable</code>	473
Table 3-672. Function <code>sdio_operation_disable</code>	474
Table 3-673. Function <code>sdio_suspend_enable</code>	474
Table 3-674. Function <code>sdio_suspend_disable</code>	475
Table 3-675. Function <code>sdio_ceata_command_enable</code>	475
Table 3-676. Function <code>sdio_ceata_command_disable</code>	476
Table 3-677. Function <code>sdio_ceata_interrupt_enable</code>	476
Table 3-678. Function <code>sdio_ceata_interrupt_disable</code>	477
Table 3-679. Function <code>sdio_ceata_command_completion_enable</code>	477
Table 3-680. Function <code>sdio_ceata_command_completion_disable</code>	478
Table 3-681. Function <code>sdio_flag_get</code>	478
Table 3-682. Function <code>sdio_flag_clear</code>	479
Table 3-683. Function <code>sdio_interrupt_enable</code>	480
Table 3-684. Function <code>sdio_interrupt_disable</code>	481
Table 3-685. Function <code>sdio_interrupt_flag_get</code>	483
Table 3-686. Function <code>sdio_interrupt_flag_clear</code>	484
Table 3-687. SPI/I2S Registers	485
Table 3-688. SPI/I2S firmware function	486
Table 3-689. Structure <code>spi_parameter_struct</code>	487
Table 3-690. Function <code>spi_i2s_deinit</code>	487
Table 3-691. Function <code>spi_struct_para_init</code>	488
Table 3-692. Function <code>spi_init</code>	488
Table 3-693. Function <code>spi_enable</code>	489
Table 3-694. Function <code>spi_disable</code>	490
Table 3-695. Function <code>i2s_init</code>	490
Table 3-696. Function <code>i2s_psc_config</code>	491
Table 3-697. Function <code>i2s_enable</code>	492
Table 3-698. Function <code>i2s_disable</code>	493

Table 3-699. Function spi_nss_output_enable	493
Table 3-700. Function spi_nss_output_disable	494
Table 3-701. Function spi_nss_internal_high	494
Table 3-702. Function spi_nss_internal_low	495
Table 3-703. Function spi_dma_enable	495
Table 3-704. Function spi_dma_disable	496
Table 3-705. Function spi_i2s_data_frame_format_config	497
Table 3-706. Function spi_bidirectional_transfer_config	497
Table 3-707. Function spi_i2s_data_transmit	498
Table 3-708. Function spi_i2s_data_receive	499
Table 3-709. Function spi_crc_polynomial_set	499
Table 3-710. Function spi_crc_polynomial_get	500
Table 3-711. Function spi_crc_on	500
Table 3-712. Function spi_crc_off	501
Table 3-713. Function spi_crc_next	501
Table 3-714. Function spi_crc_get	502
Table 3-715. Function spi_quad_enable	502
Table 3-716. Function spi_quad_disable	503
Table 3-717. Function spi_quad_write_enable	503
Table 3-718. Function spi_quad_read_enable	504
Table 3-719. Function spi_quad_io23_output_enable	504
Table 3-720. Function spi_quad_io23_output_disable	505
Table 3-721. Function spi_i2s_flag_get	505
Table 3-722. Function spi_i2s_interrupt_enable	506
Table 3-723. Function spi_i2s_interrupt_disable	507
Table 3-724. Function spi_i2s_interrupt_flag_get	508
Table 3-725. Function spi_crc_error_clear	508
Table 3-726. TIMEx Registers	509
Table 3-727. TIMEx firmware function	510
Table 3-728. Structure timer_parameter_struct	512
Table 3-729. Structure timer_break_parameter_struct	512
Table 3-730. Structure timer_oc_parameter_struct	513
Table 3-731. Structure timer_ic_parameter_struct	513
Table 3-732. Function timer_deinit	513
Table 3-733. Function timer_struct_para_init	514
Table 3-734. Function timer_init	514
Table 3-735. Function timer_enable	515
Table 3-736. Function timer_disable	516
Table 3-737. Function timer_auto_reload_shadow_enable	516
Table 3-738. Function timer_auto_reload_shadow_disable	517
Table 3-739. Function timer_update_event_enable	517
Table 3-740. Function timer_update_event_disable	518
Table 3-741. Function timer_counter_alignment	518
Table 3-742. Function timer_counter_up_direction	519

Table 3-743. Function timer_counter_down_direction	520
Table 3-744. Function timer_prescaler_config.....	520
Table 3-745. Function timer_repetition_value_config	521
Table 3-746. Function timer_autoreload_value_config	521
Table 3-747. Function timer_counter_value_config.....	522
Table 3-748. Function timer_counter_read	522
Table 3-749. Function timer_prescaler_read	523
Table 3-750. Function timer_single_pulse_mode_config	523
Table 3-751. Function timer_update_source_config.....	524
Table 3-752. Function timer_dma_enable	525
Table 3-753. Function timer_dma_disable	526
Table 3-754. Function timer_channel_dma_request_source_select.....	526
Table 3-755. Function timer_dma_transfer_config.....	527
Table 3-756. Function timer_event_software_generate.....	529
Table 3-757. Function timer_break_struct_para_init	530
Table 3-758. Function timer_break_config	530
Table 3-759. Function timer_break_enable.....	531
Table 3-760. Function timer_break_disable	531
Table 3-761. Function timer_automatic_output_enable	532
Table 3-762. Function timer_automatic_output_disable	533
Table 3-763. Function timer_primary_output_config.....	533
Table 3-764. Function timer_channel_control_shadow_config	534
Table 3-765. Function timer_channel_control_shadow_update_config.....	534
Table 3-766. Function timer_channel_output_struct_para_init	535
Table 3-767. Function timer_channel_output_config	536
Table 3-768. Function timer_channel_output_mode_config	537
Table 3-769. Function timer_channel_output_pulse_value_config.....	538
Table 3-770. Function timer_channel_output_shadow_config	538
Table 3-771. Function timer_channel_output_fast_config.....	539
Table 3-772. Function timer_channel_output_clear_config	540
Table 3-773. Function timer_channel_output_polarity_config.....	541
Table 3-774. Function timer_channel_complementary_output_polarity_config	542
Table 3-775. Function timer_channel_output_state_config	542
Table 3-776. Function timer_channel_complementary_output_state_config	543
Table 3-777. Function timer_channel_input_struct_para_init.....	544
Table 3-778. Function timer_input_capture_config.....	545
Table 3-779. Function timer_channel_input_capture_prescaler_config	545
Table 3-780. Function timer_channel_capture_value_register_read	546
Table 3-781. Function timer_input_pwm_capture_config.....	547
Table 3-782. Function timer_hall_mode_config.....	548
Table 3-783. Function timer_input_trigger_source_select	548
Table 3-784. Function timer_master_output_trigger_source_select	549
Table 3-785. Function timer_slave_mode_select	551
Table 3-786. Function timer_master_slave_mode_config	551

Table 3-787. Function timer_external_trigger_config	552
Table 3-788. Function timer_quadrature_decoder_mode_config	553
Table 3-789. Function timer_internal_clock_config	554
Table 3-790. Function timer_internal_trigger_as_external_clock_config	555
Table 3-791. Function timer_external_trigger_as_external_clock_config	555
Table 3-792. Function timer_external_clock_mode0_config	556
Table 3-793. Function timer_external_clock_mode1_config	557
Table 3-794. Function timer_external_clock_mode1_disable	558
Table 3-795. Function timer_flag_get	559
Table 3-796. Function timer_flag_clear	560
Table 3-797. Function timer_interrupt_enable	560
Table 3-798. Function timer_interrupt_disable	561
Table 3-799. Function timer_interrupt_flag_get	562
Table 3-800. Function timer_interrupt_flag_clear	563
Table 3-801. TLI Registers	564
Table 3-802. TLI firmware function	564
Table 3-803. Structure tli_parameter_struct	565
Table 3-804. Structure tli_layer_parameter_struct	566
Table 3-805. Structure tli_layer_lut_parameter_struct	566
Table 3-806. Enum tli_layer_ppf_enum	566
Table 3-807. Function tli_deinit	567
Table 3-808. Function tli_struct_para_init	567
Table 3-809. Function tli_init	568
Table 3-810. Function tli_dither_config	569
Table 3-811. Function tli_enable	570
Table 3-812. Function tli_disable	570
Table 3-813. Function tli_reload_config	570
Table 3-814. Function tli_layer_struct_para_init	571
Table 3-815. Function tli_layer_init	572
Table 3-816. Function tli_layer_window_offset_modify	573
Table 3-817. Function tli_lut_struct_para_init	574
Table 3-818. Function tli_lut_init	574
Table 3-819. Function tli_ckey_init	575
Table 3-820. Function tli_layer_enable	576
Table 3-821. Function tli_layer_disable	576
Table 3-822. Function tli_color_key_enable	577
Table 3-823. Function tli_color_key_disable	577
Table 3-824. Function tli_lut_enable	578
Table 3-825. Function tli_lut_disable	578
Table 3-826. Function tli_line_mark_set	579
Table 3-827. Function tli_current_pos_get	579
Table 3-828. Function tli_interrupt_enable	580
Table 3-829. Function tli_interrupt_disable	580
Table 3-830. Function tli_interrupt_flag_get	581

Table 3-831. Function tli_interrupt_flag_clear.....	581
Table 3-832. Function tli_flag_get	582
Table 3-833. TRNG Registers	583
Table 3-834. TRNG firmware function.....	583
Table 3-835. Enum trng_flag_enum	583
Table 3-836. Enum trng_int_flag_enum.....	584
Table 3-837. Function trng_deinit.....	584
Table 3-838. Function trng_enable.....	584
Table 3-839. Function trng_disable.....	585
Table 3-840. Function trng_get_true_random_data	585
Table 3-841. Function trng_flag_get	586
Table 3-842. Function trng_interrupt_enable	586
Table 3-843. Function trng_interrupt_disable	587
Table 3-844. Function trng_interrupt_flag_get.....	587
Table 3-845. Function trng_interrupt_flag_clear.....	588
Table 3-846. USART Registers	589
Table 3-847. USART firmware function.....	589
Table 3-848. Enum usart_flag_enum.....	590
Table 3-849. Enum usart_interrupt_flag_enum.....	591
Table 3-850. Enum usart_interrupt_enum.....	591
Table 3-851. Enum usart_invert_enum	592
Table 3-852. Function usart_deinit.....	592
Table 3-853. Function usart_baudrate_set	592
Table 3-854. Function usart_parity_config	593
Table 3-855. Function usart_word_length_set.....	594
Table 3-856. Function usart_stop_bit_set.....	594
Table 3-857. Function usart_enable	595
Table 3-858. Function usart_disable	595
Table 3-859. Function usart_transmit_config.....	596
Table 3-860. Function usart_receive_config	597
Table 3-861. Function usart_data_first_config	597
Table 3-862. Function usart_invert_config	598
Table 3-863. Function usart_receiver_timeout_enable.....	598
Table 3-864. Function usart_receiver_timeout_disable.....	599
Table 3-865. Function usart_receiver_timeout_threshold_config	599
Table 3-866. Function usart_data_transmit	600
Table 3-867. Function usart_data_receive	601
Table 3-868. Function usart_address_config	601
Table 3-869. Function usart_mute_mode_enable.....	602
Table 3-870. Function usart_mute_mode_disable.....	602
Table 3-871. Function usart_mute_mode_wakeup_config	603
Table 3-872. Function usart_lin_mode_enable	603
Table 3-873. Function usart_lin_mode_disable	604
Table 3-874. Function usart_lin_break_dection_length_config	605

Table 3-875. Function <code>usart_send_break</code>	605
Table 3-876. Function <code>usart_halfduplex_enable</code>	606
Table 3-877. Function <code>usart_halfduplex_disable</code>	606
Table 3-878. Function <code>usart_synchronous_clock_enable</code>	607
Table 3-879. Function <code>usart_synchronous_clock_disable</code>	607
Table 3-880. Function <code>usart_synchronous_clock_config</code>	608
Table 3-881. Function <code>usart_guard_time_config</code>	609
Table 3-882. Function <code>usart_smartcard_mode_enable</code>	609
Table 3-883. Function <code>usart_smartcard_mode_disable</code>	610
Table 3-884. Function <code>usart_smartcard_mode_nack_enable</code>	610
Table 3-885. Function <code>usart_smartcard_mode_nack_disable</code>	611
Table 3-886. Function <code>usart_smartcard_autoretry_config</code>	611
Table 3-887. Function <code>usart_block_length_config</code>	612
Table 3-888. Function <code>usart_irda_mode_enable</code>	612
Table 3-889. Function <code>usart_irda_mode_disable</code>	613
Table 3-890. Function <code>usart_prescaler_config</code>	613
Table 3-891. Function <code>usart_irda_lowpower_config</code>	614
Table 3-892. Function <code>usart_hardware_flow_rts_config</code>	615
Table 3-893. Function <code>usart_hardware_flow_cts_config</code>	615
Table 3-894. Function <code>usart_dma_receive_config</code>	616
Table 3-895. Function <code>usart_dma_transmit_config</code>	616
Table 3-896. Function <code>usart_flag_get</code>	617
Table 3-897. Function <code>usart_flag_clear</code>	618
Table 3-898. Function <code>usart_interrupt_enable</code>	618
Table 3-899. Function <code>usart_interrupt_disable</code>	619
Table 3-900. Function <code>usart_interrupt_flag_get</code>	620
Table 3-901. Function <code>usart_interrupt_flag_clear</code>	620
Table 3-902. WWDGT Registers	621
Table 3-903. WWDGT firmware function	622
Table 3-904. Function <code>wwdgt_deinit</code>	622
Table 3-905. Function <code>wwdgt_enable</code>	622
Table 3-906. Function <code>wwdgt_counter_update</code>	623
Table 3-907. Function <code>wwdgt_config</code>	623
Table 3-908. Function <code>wwdgt_flag_get</code>	624
Table 3-909. Function <code>wwdgt_flag_clear</code>	625
Table 3-910. Function <code>wwdgt_interrupt_enable</code>	625
Table 3-911. Function <code>wwdgt_interrupt_flag_get</code>	626
Table 4-1. Revision history	627

1. Introduction

This manual introduces firmware library of GD32F20x 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 GD32F20x 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
BKP	Backup registers
CAN	Controller area network
CAU	Cryptographic Acceleration Unit
CRC	CRC calculation unit

Peripherals	Descriptions
DAC	Digital-to-analog converter
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/AFIO	General-purpose and alternate-function I/Os
HAU	Hash Acceleration Unit
I2C	Inter-integrated circuit interface
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
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 “gd32f20x_”, such as: gd32f20x_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 uppcase of English letters;
- Registers are handled as constants. The naming of them are written in uppcase 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;

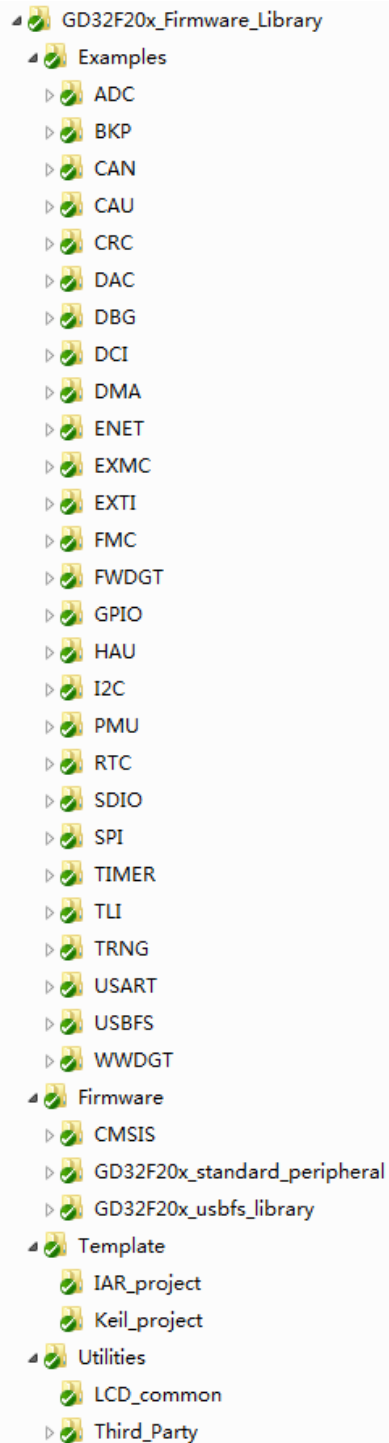
- The naming of peripheral functions are started with the peripheral abbreviation added with an underline, when the function name is concluded by several words, underlines should be adapted among words, and all the peripheral functions are written in lowercase.

2. Firmware Library Overview

2.1. File Structure of Firmware Library

GD32F20x_Firmware_Library, the file structure is shown as below:

Figure 2-1. File structure of firmware library of GD32F20x



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;
- `gd32f20x_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);
- `gd32f20x_it.c`: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- `gd32f20x_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 M3 kernel support files, the startup file based on the Cortex M3 kernel processor, the global header file of GD32F20x and system configuration file;
- GD32F20x_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;
- GD32F20x_usbfs_driver subfolder includes all the related files about 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.

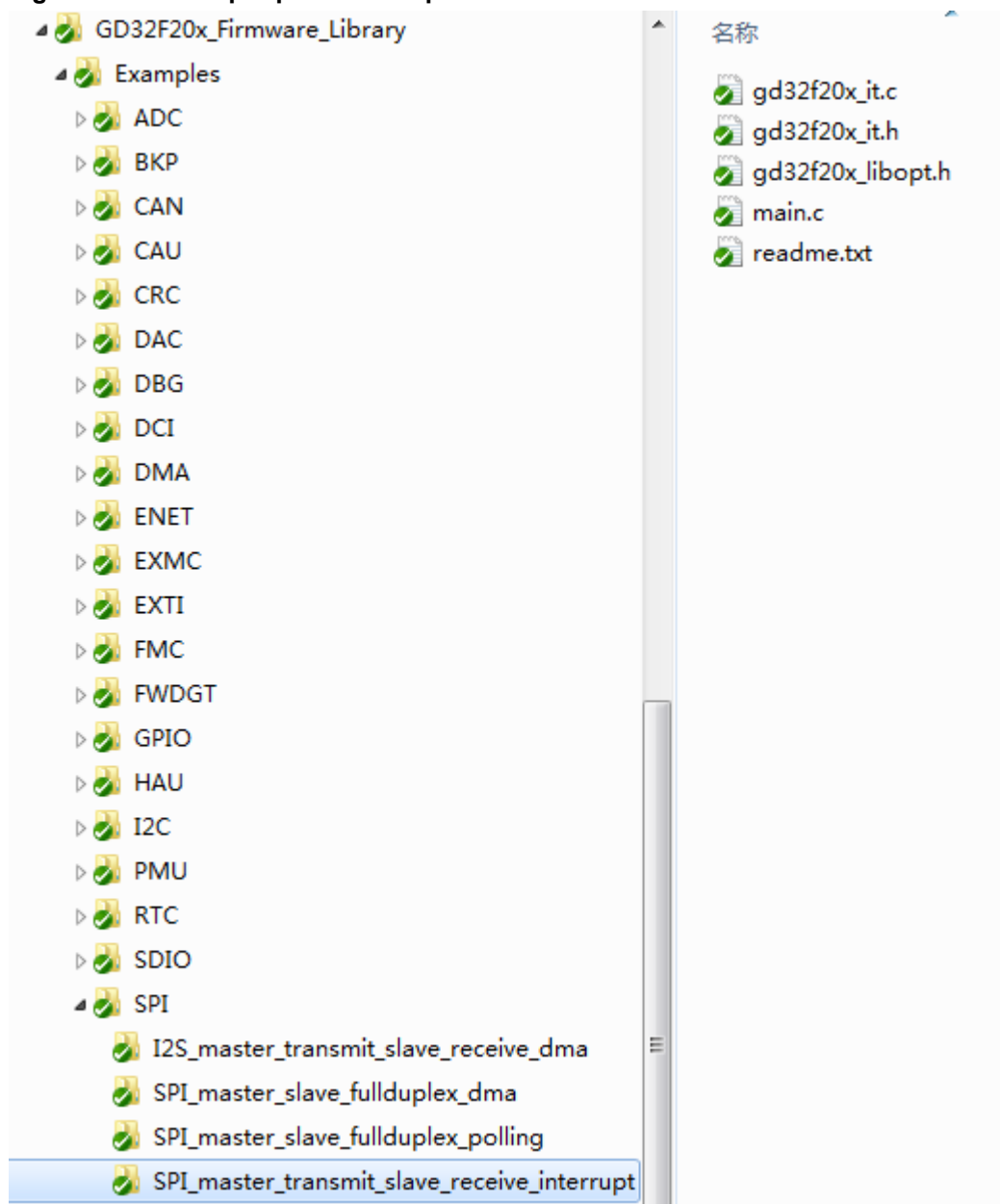
2.1.3. Template Folder

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 formware 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:

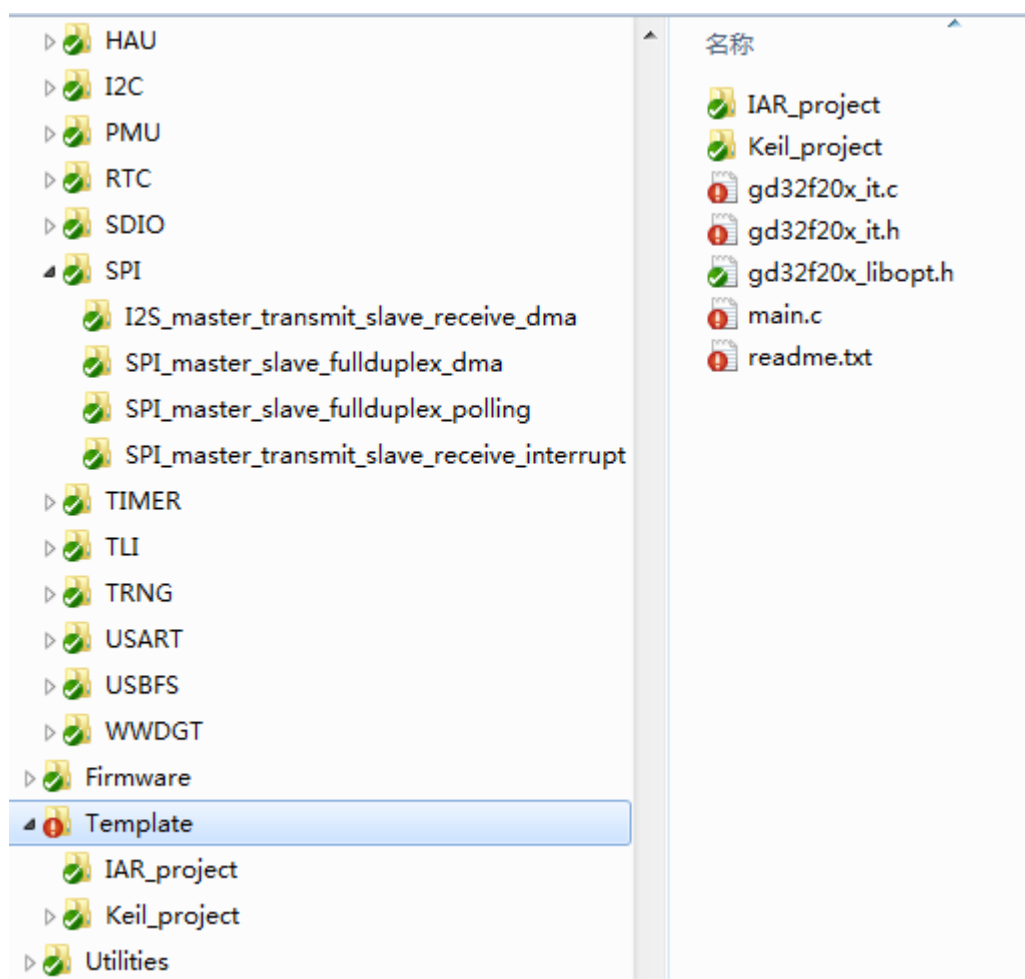
Figure 2-2. Select peripheral example files



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:

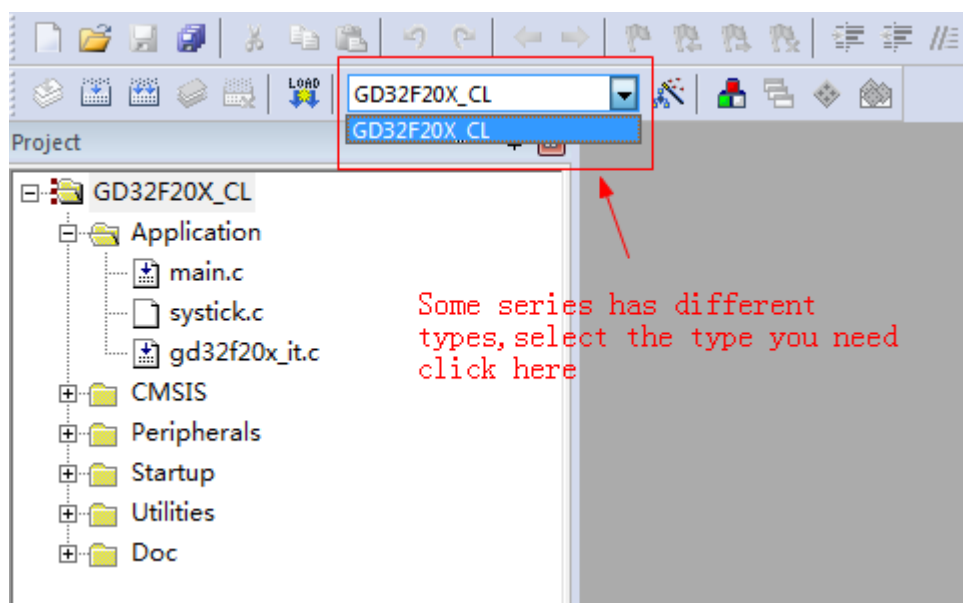
Figure 2-3. Copy the peripheral example files



Open a project

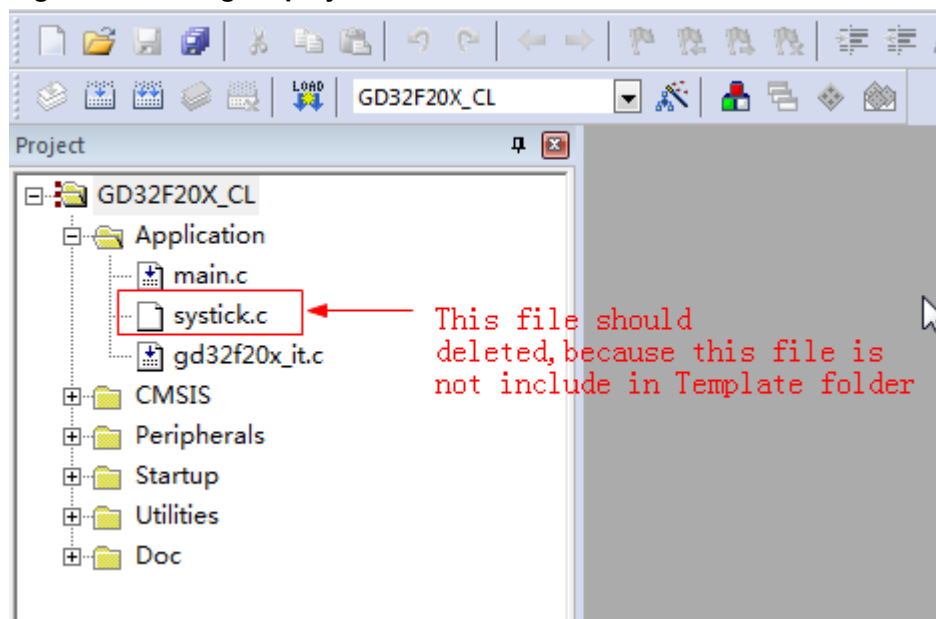
GD provides project in Keil and IAR, users can open project in different IDEs according to their need, such as “Keil_project”, open \Template\Keil_project\Project.uvproj, shown as below:

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:

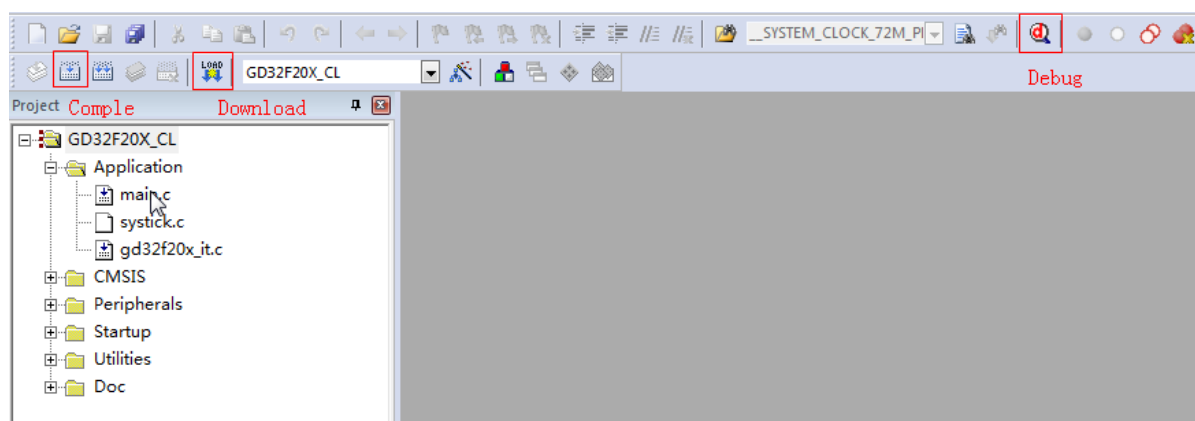
Figure 2-5. Configure project files



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



2.1.4. Utilities Folder

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

- LCD_Commom and Third_Party subfolders include files for USB tests;
- gd32f20x_eval.h and gd32f20x_lcd_eval.h are related header files of the evaluation board about running the firmware examples;
- gd32f20x_eval.c and gd32f20x_lcd_eval.c are 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
gd32f20x_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.
gd32f20x_it.h	Header file, including all the prototypes of interrupt service routines.
gd32f20x_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.

gd32f20x_xxx.h	The header file of peripheral PPP, including functions about peripheral PPP, and the variables used for functions.
gd32f20x_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

Registers	Descriptions
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
ADC_WDHT	watchdog high threshold register
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
ADC_RDATA	regular data register
ADC_OVSAMPCTL	oversample control 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_mode_config	configure the ADC mode(only for ADC0)
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_tempsensor_vrefint_enable	enable the temperature sensor and Vrefint channel
adc_tempsensor_vrefint_disable	disable the temperature sensor and Vrefint channel
adc_dma_mode_enable	enable DMA request
adc_dma_mode_disable	disable DMA request
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
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_regular_data_read	read ADC regular group data register

Function name	Function description
adc_inserted_data_read	read ADC inserted group data register
adc_sync_mode_convert_value_read	read the last ADC0 and ADC1 conversion result data in sync mode
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_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_flag_get	get the ADC flag bits
adc_flag_clear	clear the ADC flag bits
adc_interrupt_enable	enable ADC interrupt
adc_interrupt_disable	disable ADC interrupt
adc_interrupt_flag_get	get the ADC interrupt bits
adc_interrupt_flag_clear	clear the ADC flag

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(uint32_t adc_periph);
Function descriptions	reset ADCx peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
adc_periph	ADC peripheral
ADCx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset ADC0 */
adc_deinit(ADC0);
```

adc_mode_config

The description of adc_mode_config is shown as below:

Table 3-5. Function adc_mode_config

Function name	adc_mode_config
Function prototype	void adc_mode_config(uint32_t mode);
Function descriptions	configure the ADCs sync mode
Precondition	-
The called functions	-
Input parameter{in}	
mode	ADC mode
ADC_MODE_FREE	all the ADC work independently
ADC_DAUL_REGULAL _PARALLEL_INSERTE D_PARALLEL	ADC0 and ADC1 work in combined regular parallel + inserted parallel mode
ADC_DAUL_REGULAL _PARALLEL_INSERTE D_ROTATION	ADC0 and ADC1 work in combined regular parallel + trigger rotation mode
ADC_DAUL_INSERTE D_PARALLEL_REGUL AL_FOLLOWUP_FAST	ADC0 and ADC1 work in combined inserted parallel + follow-up fast mode
ADC_DAUL_INSERTE D_PARALLEL_REGUL AL_FOLLOWUP_SLO W	ADC0 and ADC1 work in combined inserted parallel + follow-up slow mode
ADC_DAUL_INSERTE D_PARALLEL	ADC0 and ADC1 work in inserted parallel mode only
ADC_DAUL_REGULAL _PARALLEL	ADC0 and ADC1 work in regular parallel mode only
ADC_DAUL_REGULAL _FOLLOWUP_FAST	ADC0 and ADC1 work in follow-up fast mode only
ADC_DAUL_REGULAL _FOLLOWUP_SLOW	ADC0 and ADC1 work in follow-up slow mode only
ADC_DAUL_INSERTE D_TRRIGGER_ROTAT ION	ADC0 and ADC1 work in trigger rotation mode only
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the ADC sync mode */
```



```
adc_mode_config(ADC_MODE_FREE);
```

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	<code>adc_special_function_config</code>
Function prototype	<code>void adc_special_function_config(uint32_t adc_periph, uint32_t function, ControlStatus new_value);</code>
Function descriptions	enable or disable ADC special function
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx</i>	x=0,1,2
Input parameter{in}	
function	the function to config
<i>ADC_SCAN_MODE</i>	scan mode select
<i>ADC_INSERTED_CHANNEL_AUTO</i>	inserted channel group convert automatically
<i>ADC_CONTINUOUS_MODE</i>	continuous mode select
Input parameter{in}	
newvalue	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</i>	x=0,1,2
Input parameter{in}	
data_alignment	data alignment select
<i>ADC_DATAALIGN_RIGHT</i>	LSB alignment
<i>ADC_DATAALIGN_LEFT</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}	
adc_periph	ADC peripheral
<i>ADCx</i>	x=0,1,2
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=0,1,2
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
ADCx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

adc_tempsensor_vrefint_enable

The description of adc_tempsensor_vrefint_enable is shown as below:

Table 3-11. Function adc_tempsensor_vrefint_enable

Function name	adc_tempsensor_vrefint_enable
Function prototype	void adc_tempsensor_vrefint_enable(void);
Function descriptions	enable the temperature sensor and Vrefint channel
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the temperature sensor and Vrefint channel */
adc_tempsensor_vrefint_enable();
```

adc_tempsensor_vrefint_disable

The description of adc_tempsensor_vrefint_disable is shown as below:

Table 3-12. Function adc_tempsensor_vrefint_disable

Function name	adc_tempsensor_vrefint_disable
Function prototype	void adc_tempsensor_vrefint_disable(void);
Function descriptions	disable the temperature sensor and Vrefint channel
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the temperature sensor and Vrefint channel */
adc_tempsensor_vrefint_disable();
```

adc_dma_mode_enable

The description of adc_dma_mode_enable is shown as below:

Table 3-13. 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</i>	x=0,1,2
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 3-14. 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</i>	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 DMA request */
adc_dma_mode_disable(ADC0);
```

adc_discontinuous_mode_config

The description of adc_discontinuous_mode_config is shown as below:

Table 3-15. 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
<i>ADCx</i>	x=0,1,2
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
<i>ADC_CHANNEL_DISCON_DISABLE</i>	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 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 3-16. 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
<i>ADCx</i>	x=0,1,2
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}	
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 3-17. 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
<i>ADCx</i>	x=0,1,2
Input parameter{in}	
rank	the regular group sequence rank, this parameter must be from 0 to 15
Input parameter{in}	
adc_channel	the selected ADC channel
<i>ADC_CHANNEL_x</i>	ADC Channelx (x=0...17)(x=16 and x=17 are only for ADC0)
Input parameter{in}	
sample_time	the sample time value
<i>ADC_SAMPLETIME_1</i>	1.5 cycles

<i>POINT5</i>	
<i>ADC_SAMPLETIME_7</i> <i>POINT5</i>	7.5 cycles
<i>ADC_SAMPLETIME_1</i> <i>3POINT5</i>	13.5 cycles
<i>ADC_SAMPLETIME_2</i> <i>8POINT5</i>	28.5 cycles
<i>ADC_SAMPLETIME_4</i> <i>1POINT5</i>	41.5 cycles
<i>ADC_SAMPLETIME_5</i> <i>5POINT5</i>	55.5 cycles
<i>ADC_SAMPLETIME_7</i> <i>1POINT5</i>	71.5 cycles
<i>ADC_SAMPLETIME_2</i> <i>39POINT5</i>	239.5 cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 regular channel */
```

```
adc_regular_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_7POINT5);
```

adc_inserted_channel_config

The description of `adc_inserted_channel_config` is shown as below:

Table 3-18. Function `adc_inserted_channel_config`

Function name	<code>adc_inserted_channel_config</code>
Function prototype	<code>void adc_inserted_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);</code>
Function descriptions	configure ADC inserted channel
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx</i>	x=0,1,2
Input parameter{in}	
rank	the inserted group sequencer rank, this parameter must be from 0 to 3
Input parameter{in}	
adc_channel	the selected ADC channel
<i>ADC_CHANNEL_x</i>	ADC Channelx (x=0...17)(x=16 and x=17 are only for ADC0)

Input parameter{in}	
sample_time	the sample time value
ADC_SAMPLETIME_1 POINT5	1.5 cycles
ADC_SAMPLETIME_7 POINT5	7.5 cycles
ADC_SAMPLETIME_1 3POINT5	13.5 cycles
ADC_SAMPLETIME_2 8POINT5	28.5 cycles
ADC_SAMPLETIME_4 1POINT5	41.5 cycles
ADC_SAMPLETIME_5 5POINT5	55.5 cycles
ADC_SAMPLETIME_7 1POINT5	71.5 cycles
ADC_SAMPLETIME_2 39POINT5	239.5 cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 inserted channel */
```

```
adc_inserted_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_7POINT5);
```

adc_inserted_channel_offset_config

The description of adc_inserted_channel_offset_config is shown as below:

Table 3-19. 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
ADCx	x=0,1,2
Input parameter{in}	
inserted_channel	insert channel select

<i>ADC_INSERTED_CHANNEL_NNEL_x</i>	inserted channel, x=0,1,2,3
Input parameter{in}	
offset	the offset data, this parameter must be from 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 3-20. Function `adc_external_trigger_source_config`

Function name	<code>adc_external_trigger_source_config</code>
Function prototype	<code>void adc_external_trigger_source_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t external_trigger_source);</code>
Function descriptions	configure ADC external trigger source
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx</i>	x=0,1,2
Input parameter{in}	
adc_channel_group	select the channel group
<i>ADC_REGULAR_CHANNEL_NNEL</i>	regular channel group
<i>ADC_INSERTED_CHANNEL_NNEL</i>	inserted channel group
Input parameter{in}	
external_trigger_source	regular or inserted group trigger source
<i>ADC0_1_EXTTRIGGER_REGULAR_T0_CH0</i>	TIMER0 CH0 event select for regular channel
<i>ADC0_1_EXTTRIGGER_REGULAR_T0_CH1</i>	TIMER0 CH1 event select for regular channel
<i>ADC0_1_EXTTRIGGER_REGULAR_T0_CH2</i>	TIMER0 CH2 event select for regular channel
<i>ADC0_1_EXTTRIGGER_REGULAR_T1_CH1</i>	TIMER1 CH1 event select for regular channel

<i>GULAR_T1_CH1</i>	
<i>ADC0_1_EXTTRIG_REGULAR_T2_TRGO</i>	TIMER2 TRGO event select for regular channel
<i>ADC0_1_EXTTRIG_REGULAR_T3_CH3</i>	TIMER3 CH3 event select for regular channel
<i>ADC0_1_EXTTRIG_REGULAR_T7_TRGO</i>	TIMER7 TRGO event select for regular channel
<i>ADC0_1_EXTTRIG_REGULAR_EXTI_11</i>	external interrupt line 11 for regular channel
<i>ADC2_EXTTRIG_REGULAR_T2_CH0</i>	TIMER2 CH0 event select for regular channel
<i>ADC2_EXTTRIG_REGULAR_T1_CH2</i>	TIMER1 CH2 event select for regular channel
<i>ADC2_EXTTRIG_REGULAR_T0_CH2</i>	TIMER0 CH2 event select for regular channel
<i>ADC2_EXTTRIG_REGULAR_T7_CH0</i>	TIMER7 CH0 event select for regular channel
<i>ADC2_EXTTRIG_REGULAR_T7_TRGO</i>	TIMER7 TRGO event select for regular channel
<i>ADC2_EXTTRIG_REGULAR_T4_CH0</i>	TIMER4 CH0 event select for regular channel
<i>ADC2_EXTTRIG_REGULAR_T4_CH2</i>	TIMER4 CH2 event select for regular channel
<i>ADC0_1_2_EXTTRIG_REGULAR_NONE</i>	software trigger for regular channel
<i>ADC0_1_EXTTRIG_INSERTED_T0_TRGO</i>	TIMER0 TRGO event select for inserted channel
<i>ADC0_1_EXTTRIG_INSERTED_T0_CH3</i>	TIMER0 CH3 event select for inserted channel
<i>ADC0_1_EXTTRIG_INSERTED_T1_TRGO</i>	TIMER1 TRGO event select for inserted channel
<i>ADC0_1_EXTTRIG_INSERTED_T1_CH0</i>	TIMER1 CH0 event select for inserted channel
<i>ADC0_1_EXTTRIG_INSERTED_T2_CH3</i>	TIMER2 CH3 event select for inserted channel
<i>ADC0_1_EXTTRIG_INSERTED_T3_TRGO</i>	TIMER3 TRGO event select for inserted channel
<i>ADC0_1_EXTTRIG_INSERTED_EXTI_15</i>	external interrupt line 15 for inserted channel
<i>ADC0_1_EXTTRIG_INSERTED_T7_CH3</i>	TIMER7 CH3 event select for inserted channel
<i>ADC2_EXTTRIG_INSERTED_T0_TRGO</i>	TIMER0 TRGO event select for inserted channel

<code>ADC2_EXTTRIG_INSE RTED_T0_CH3</code>	TIMER0 CH3 event select for inserted channel
<code>ADC2_EXTTRIG_INSE RTED_T3_CH2</code>	TIMER3 CH2 event select for inserted channel
<code>ADC2_EXTTRIG_INSE RTED_T7_CH1</code>	TIMER7 CH1 event select for inserted channel
<code>ADC2_EXTTRIG_INSE RTED_T7_CH3</code>	TIMER7 CH3 event select for inserted channel
<code>ADC2_EXTTRIG_INSE RTED_T4_TRGO</code>	TIMER4 TRGO event select for inserted channel
<code>ADC2_EXTTRIG_INSE RTED_T4_CH3</code>	TIMER4 CH3 event select for inserted channel
<code>ADC0_1_2_EXTTRIG_I NSERTED_NONE</code>	software trigger 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,  
ADC0_1_EXTTRIG_REGULAR_T0_CH0);
```

adc_external_trigger_config

The description of `adc_external_trigger_config` is shown as below:

Table 3-21. Function `adc_external_trigger_config`

Function name	<code>adc_external_trigger_config</code>
Function prototype	<code>void adc_external_trigger_config(uint32_t adc_periph, uint8_t adc_channel_group, ControlStatus newvalue);</code>
Function descriptions	configure ADC external trigger
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<code>ADCx</code>	x=0,1,2
Input parameter{in}	
adc_channel_group	select the channel group
<code>ADC_REGULAR_CHA NNEL</code>	regular channel group
<code>ADC_INSERTED_CHA</code>	inserted channel group

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

Example:

```
/* enable ADC0 inserted channel group external trigger */
```

```
adc_external_trigger_config(ADC0, ADC_INSERTED_CHANNEL_0, ENABLE);
```

adc_software_trigger_enable

The description of adc_software_trigger_enable is shown as below:

Table 3-22. 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</i>	x=0,1,2
Input parameter{in}	
adc_channel_group	select the channel group
<i>ADC_REGULAR_CHA</i> <i>NNEL</i>	regular channel group
<i>ADC_INSERTED_CHA</i> <i>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_regular_data_read

The description of adc_regular_data_read is shown as below:

Table 3-23. 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</i>	x=0,1,2
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 3-24. 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}	
adc_periph	ADC peripheral
<i>ADCx</i>	x=0,1,2
Input parameter{in}	
inserted_channel	insert channel select
<i>ADC_INSERTED_CHANNEL_x</i>	inserted Channelx, x=0,1,2,3
Output parameter{out}	
-	-

Return value	
uint16_t	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_sync_mode_convert_value_read

The description of adc_sync_mode_convert_value_read is shown as below:

Table 3-25. Function adc_sync_mode_convert_value_read

Function name	adc_sync_mode_convert_value_read
Function prototype	uint32_t adc_sync_mode_convert_value_read(void);
Function descriptions	read the last ADC0 and ADC1 conversion result data in sync mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	ADC conversion value (0-0xFFFFFFFF)

Example:

```
/* read the last ADC0 and ADC1 conversion result data in sync mode */
```

```
uint32_t adc_value = 0;
```

```
adc_value = adc_sync_mode_convert_value_read();
```

adc_watchdog_single_channel_enable

The description of adc_watchdog_single_channel_enable is shown as below:

Table 3-26. 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

ADCx	x=0,1,2
Input parameter{in}	
adc_channel	the selected ADC channel
ADC_CHANNEL_x	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 3-27. 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
ADCx	x=0,1,2
Input parameter{in}	
adc_channel_group	the channel group use analog watchdog
ADC_REGULAR_CHANNEL	regular channel group
ADC_INSERTED_CHANNEL	inserted channel group
ADC_REGULAR_INSERTED_CHANNEL	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 3-28. 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}	
adc_periph	ADC peripheral
ADCx	x=0,1,2
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 3-29. 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}	
adc_periph	ADC peripheral
ADCx	x=0,1,2
Input parameter{in}	
low_threshold	analog watchdog low threshold, 0...4095
Input parameter{in}	
high_threshold	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_resolution_config

The description of adc_resolution_config is shown as below:

Table 3-30. 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
ADCx	x=0,1,2
Input parameter{in}	
resolution	ADC resolution
ADC_RESOLUTION_12B	12-bit ADC resolution
ADC_RESOLUTION_10B	10-bit ADC resolution
ADC_RESOLUTION_8B	8-bit ADC resolution
ADC_RESOLUTION_6B	6-bit ADC resolution
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config ADC0 resolution */
```

```
adc_resolution_config(ADC0, ADC_RESOLUTION_10B);
```

adc_oversample_mode_config

The description of adc_oversample_mode_config is shown as below:

Table 3-31. Function adc_oversample_mode_config

Function name	adc_oversample_mode_config
Function prototype	void adc_oversample_mode_config(uint32_t adc_periph, uint8_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=0,1,2
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
ADC_OVERSAMPLING _SHIFT_3B	3-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_4B	4-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_5B	5-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_6B	6-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_7B	7-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_8B	8-bit oversampling shift
Input parameter{in}	
ratio	ADC oversampling ratio
ADC_OVERSAMPLING _RATIO_MUL2	oversampling ratio multiple 2
ADC_OVERSAMPLING _RATIO_MUL4	oversampling ratio multiple 4
ADC_OVERSAMPLING _RATIO_MUL8	oversampling ratio multiple 8
ADC_OVERSAMPLING _RATIO_MUL16	oversampling ratio multiple 16

ADC_OVERSAMPLING_RATIO_MUL32	oversampling ratio multiple 32
ADC_OVERSAMPLING_RATIO_MUL64	oversampling ratio multiple 64
ADC_OVERSAMPLING_RATIO_MUL128	oversampling ratio multiple 128
ADC_OVERSAMPLING_RATIO_MUL256	oversampling ratio multiple 256
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config ADC0 oversample mode */
```

```
adc_oversample_mode_config (ADC0, ADC_OVERSAMPLING_ALL_CONVERT,
ADC_OVERSAMPLING_SHIFT_2B, ADC_OVERSAMPLING_RATIO_MUL4);
```

adc_oversample_mode_enable

The description of adc_oversample_mode_enable is shown as below:

Table 3-32. 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=0,1,2
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 3-33. Function `adc_oversample_mode_disable`

Function name	<code>adc_oversample_mode_disable</code>
Function prototype	<code>void adc_oversample_mode_disable(uint32_t adc_periph);</code>
Function descriptions	disable ADC oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx</i>	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

adc_flag_get

The description of `adc_flag_get` is shown as below:

Table 3-34. Function `adc_flag_get`

Function name	<code>adc_flag_get</code>
Function prototype	<code>FlagStatus adc_flag_get(uint32_t adc_periph , uint32_t adc_flag);</code>
Function descriptions	get the ADC flag
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx</i>	x=0,1,2
Input parameter{in}	
adc_flag	the adc flag
<i>ADC_FLAG_WDE</i>	analog watchdog event flag
<i>ADC_FLAG_EOC</i>	end of group conversion flag
<i>ADC_FLAG_EOIC</i>	end of inserted group conversion flag
<i>ADC_FLAG_STIC</i>	start flag of inserted channel group
<i>ADC_FLAG_STRC</i>	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 3-35. 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
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx</i>	x=0,1,2
Input parameter{in}	
adc_flag	the adc flag
<i>ADC_FLAG_WDE</i>	analog watchdog event flag
<i>ADC_FLAG_EOC</i>	end of group conversion flag
<i>ADC_FLAG_EOIC</i>	end of inserted group conversion flag
<i>ADC_FLAG_STIC</i>	start flag of inserted channel group
<i>ADC_FLAG_STRC</i>	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_interrupt_enable

The description of adc_interrupt_enable is shown as below:

Table 3-36. 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
<i>ADCx</i>	x=0,1,2
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:

```
/* 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 3-37. Function `adc_interrupt_disable`

Function name	<code>adc_interrupt_disable</code>
Function prototype	<code>void adc_interrupt_disable(uint32_t adc_periph , uint32_t adc_interrupt);</code>
Function descriptions	disable ADC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx</i>	x=0,1,2
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_interrupt_flag_get

The description of adc_interrupt_flag_get is shown as below:

Table 3-38. 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
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx</i>	x=0,1,2
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	
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 3-39. 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
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx</i>	x=0,1,2
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:

```
/* clear the ADC0 analog watchdog interrupt bits */
```

```
adc_interrupt_flag_clear(ADC0, ADC_INT_WDE);
```

3.3. BKP

The Backup registers are located in the Backup domain that remains powered-on by V_{BAT} even if V_{DD} power is shut down, they are forty two 16-bit (84 bytes) registers for data protection of user application data, and the wake-up action from Standby mode or system reset do not affect these registers. The BKP registers are listed in chapter [3.3.1](#), the BKP firmware functions are introduced in chapter [3.3.2](#).

3.3.1. Descriptions of Peripheral registers

BKP registers are listed in the table shown as below:

Table 3-40. BKP Registers

Registers	Descriptions
BKP_DATAx (x= 0..41)	BKP data register x
BKP_OCTL	RTC signal output control register
BKP_TPCTL0	tamper pin control register0
BKP_TPCS	tamper control and status register
BKP_TPCTL1	tamper pin control register1

3.3.2. Descriptions of Peripheral functions

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

Table 3-41. BKP firmware function

Function name	Function description
bkp_deinit	reset BKP registers
bkp_data_write	write BKP data register

Function name	Function description
bkp_data_read	read BKP data register
bkp_rtc_calibration_output_enable	enable RTC clock calibration output
bkp_rtc_calibration_output_disable	disable RTC clock calibration output
bkp_rtc_signal_output_enable	enable RTC alarm or second signal output
bkp_rtc_signal_output_disable	disable RTC alarm or second signal output
bkp_rtc_output_select	select RTC output
bkp_rtc_clock_output_select	RTC clock output selection
bkp_rtc_clock_calibration_direction	RTC clock calibration direction
bkp_rtc_calibration_value_set	set RTC clock calibration value
bkp_tamper_detection_enable	enable tamper pin detection
bkp_tamper_detection_disable	disable tamper pin detection
bkp_tamper_active_level_set	set tamper pin active level
bkp_waveform_detect_config	waveform detect configure
bkp_flag_get	get BKP flag
bkp_flag_clear	clear BKP flag
bkp_tamper_interrupt_enable	enable tamper interrupt
bkp_tamper_interrupt_disable	disable tamper interrupt
bkp_interrupt_flag_get	get BKP interrupt flag
bkp_interrupt_flag_clear	clear BKP interrupt flag

Enum bkp_data_register_enum

Table 3-42. Enum bkp_data_register_enum

Member name	Function description
BKP_DATA_0	bkp data register 0
BKP_DATA_1	bkp data register 1
BKP_DATA_2	bkp data register 2
BKP_DATA_3	bkp data register 3
BKP_DATA_4	bkp data register 4
BKP_DATA_5	bkp data register 5
BKP_DATA_6	bkp data register 6
BKP_DATA_7	bkp data register 7
BKP_DATA_8	bkp data register 8
BKP_DATA_9	bkp data register 9
BKP_DATA_10	bkp data register 10
BKP_DATA_11	bkp data register 11
BKP_DATA_12	bkp data register 12
BKP_DATA_13	bkp data register 13
BKP_DATA_14	bkp data register 14
BKP_DATA_15	bkp data register 15
BKP_DATA_16	bkp data register 16
BKP_DATA_17	bkp data register 17

Member name	Function description
BKP_DATA_18	bkp data register 18
BKP_DATA_19	bkp data register 19
BKP_DATA_20	bkp data register 20
BKP_DATA_21	bkp data register 21
BKP_DATA_22	bkp data register 22
BKP_DATA_23	bkp data register 23
BKP_DATA_24	bkp data register 24
BKP_DATA_25	bkp data register 25
BKP_DATA_26	bkp data register 26
BKP_DATA_27	bkp data register 27
BKP_DATA_28	bkp data register 28
BKP_DATA_29	bkp data register 29
BKP_DATA_30	bkp data register 30
BKP_DATA_31	bkp data register 31
BKP_DATA_32	bkp data register 32
BKP_DATA_33	bkp data register 33
BKP_DATA_34	bkp data register 34
BKP_DATA_35	bkp data register 35
BKP_DATA_36	bkp data register 36
BKP_DATA_37	bkp data register 37
BKP_DATA_38	bkp data register 38
BKP_DATA_39	bkp data register 39
BKP_DATA_40	bkp data register 40
BKP_DATA_41	bkp data register 41

Enum bkp_tamper_enum

Table 3-43. Enum bkp_tamper_enum

Member name	Function description
TAMPER_0	BKP tamper0
TAMPER_1	BKP tamper1

bkp_deinit

The description of bkp_deinit is shown as below:

Table 3-44. Function bkp_deinit

Function name	bkp_deinit
Function prototype	void bkp_deinit(void);
Function descriptions	reset BKP registers
Precondition	-
The called functions	rcu_bkp_reset_enable / rcu_bkp_reset_disable
Input parameter{in}	

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

Example:

```
/* reset BKP registers */
```

```
bkp_deinit();
```

bkp_data_write

The description of bkp_data_write is shown as below:

Table 3-45. Function bkp_data_write

Function name	bkp_data_write
Function prototype	void bkp_data_write(bkp_data_register_enum register_number, uint16_t data);
Function descriptions	write BKP data register
Precondition	-
The called functions	-
Input parameter{in}	
register_number	refer to Table 3-42. Enum bkp_data_register_enum
<i>BKP_DATA_x</i> (<i>x</i> = 0..41)	bkp data register number <i>x</i>
Input parameter{in}	
data	the data to be write in BKP data register
<i>0x0000-0xFFFF</i>	data value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write BKP data register */
```

```
bkp_data_write(BKP_DATA_0, 0x1226);
```

bkp_data_read

The description of bkp_data_read is shown as below:

Table 3-46. Function bkp_data_read

Function name	bkp_data_read
Function prototype	uint16_t bkp_data_read(bkp_data_register_enum register_number);

Function descriptions	read BKP data register
Precondition	-
The called functions	-
Input parameter{in}	
register_number	refer to Table 3-42. Enum bkp_data_register_enum
<i>BKP_DATA_x</i> (<i>x</i> = 0..41)	bkp data register number <i>x</i>
Output parameter{out}	
-	-
Return value	
uint16_t	data of BKP data register(0x0000-0xFFFF)

Example:

```
/* read BKP data register0 */

uint16_t data;

data = bkp_data_read(BKP_DATA_0);
```

bkp_rtc_calibration_output_enable

The description of bkp_rtc_calibration_output_enable is shown as below:

Table 3-47. Function bkp_rtc_calibration_output_enable

Function name	bkp_rtc_calibration_output_enable
Function prototype	void bkp_rtc_calibration_output_enable(void);
Function descriptions	enable RTC clock calibration output
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable RTC clock calibration output */

bkp_rtc_calibration_output_enable();
```

bkp_rtc_calibration_output_disable

The description of bkp_rtc_calibration_output_disable is shown as below:

Table 3-48. Function bkp_rtc_calibration_output_disable

Function name	bkp_rtc_calibration_output_disable
Function prototype	void bkp_rtc_calibration_output_disable(void);
Function descriptions	disable RTC clock calibration output
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RTC clock calibration output */
```

```
bkp_rtc_calibration_output_disable();
```

bkp_rtc_signal_output_enable

The description of bkp_rtc_signal_output_enable is shown as below:

Table 3-49. Function bkp_rtc_signal_output_enable

Function name	bkp_rtc_signal_output_enable
Function prototype	void bkp_rtc_signal_output_enable (void);
Function descriptions	enable RTC alarm or second signal output
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable RTC alarm or second signal output */
```

```
bkp_rtc_signal_output_enable();
```

bkp_rtc_signal_output_disable

The description of bkp_rtc_signal_output_disable is shown as below:

Table 3-50. Function bkp_rtc_signal_output_disable

Function name	bkp_rtc_signal_output_disable
----------------------	-------------------------------

Function prototype	void bkp_rtc_signal_output_disable (void);
Function descriptions	disable RTC alarm or second signal output
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RTC alarm or second signal output */
```

```
bkp_rtc_signal_output_disable();
```

bkp_rtc_output_select

The description of bkp_rtc_output_select is shown as below:

Table 3-51. Function bkp_rtc_output_select

Function name	bkp_rtc_output_select
Function prototype	void bkp_rtc_output_select (uint16_t outputsel);
Function descriptions	select RTC output
Precondition	-
The called functions	-
Input parameter{in}	
outputsel	RTC output selection
<i>RTC_OUTPUT_ALARM_PULSE</i>	RTC alarm pulse is selected as the RTC output
<i>RTC_OUTPUT_SECOND_PULSE</i>	RTC second pulse is selected as the RTC output
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select RTC alarm pulse as the RTC output */
```

```
bkp_rtc_output_select(RTC_OUTPUT_ALARM_PULSE);
```

bkp_rtc_clock_output_select

The description of bkp_rtc_clock_output_select is shown as below:

Table 3-52. Function bkp_rtc_clock_output_select

Function name	bkp_rtc_clock_output_select
Function prototype	void bkp_rtc_clock_output_select(uint16_t clocksel);
Function descriptions	RTC clock output selection
Precondition	-
The called functions	-
Input parameter{in}	
clocksel	RTC clock output selection
<i>RTC_CLOCK_DIV64</i>	RTC clock div 64
<i>RTC_CLOCK_DIV1</i>	RTC clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* RTC clock output selection */
```

```
bkp_rtc_clock_output_select(RTC_CLOCK_DIV1);
```

bkp_rtc_clock_calibration_direction

The description of bkp_rtc_clock_calibration_direction is shown as below:

Table 3-53. Function bkp_rtc_clock_calibration_direction

Function name	bkp_rtc_clock_calibration_direction
Function prototype	void bkp_rtc_clock_calibration_direction(uint16_t direction);
Function descriptions	RTC clock calibration direction
Precondition	-
The called functions	-
Input parameter{in}	
direction	RTC clock calibration direction
<i>RTC_CLOCK_SLOW_DOWN</i>	RTC clock slow down
<i>RTC_CLOCK_SPEED_UP</i>	RTC clock speed up
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set RTC clock speed up */
```

```
bkp_rtc_clock_calibration_direction(RTC_CLOCK_SPEED_UP);
```


bkp_rtc_calibration_value_set

The description of bkp_rtc_calibration_value_set is shown as below:

Table 3-54. Function bkp_rtc_calibration_value_set

Function name	bkp_rtc_calibration_value_set
Function prototype	void bkp_rtc_calibration_value_set(uint8_t value);
Function descriptions	set RTC clock calibration value
Precondition	-
The called functions	-
Input parameter{in}	
value	RTC clock calibration value
0x00 - 0x7F	value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set RTC clock calibration value */
bkp_rtc_calibration_value_set(0x7F);
```

bkp_tamper_detection_enable

The description of bkp_tamper_detection_enable is shown as below:

Table 3-55. Function bkp_tamper_detection_enable

Function name	bkp_tamper_detection_enable
Function prototype	void bkp_tamper_detection_enable(bkp_tamper_enum tamperx);
Function descriptions	enable tamper pin detection
Precondition	-
The called functions	-
Input parameter{in}	
tamperx	refer to Table 3-43. Enum bkp_tamper_enum
TAMPER_0	BKP tamper0
TAMPER_1	BKP tamper1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable tamper0 pin detection */
bkp_tamper_detection_enable(TAMPER_0);
```

bkp_tamper_detection_disable

The description of bkp_tamper_detection_disable is shown as below:

Table 3-56. Function bkp_tamper_detection_disable

Function name	bkp_tamper_detection_disable
Function prototype	void bkp_tamper_detection_disable(bkp_tamper_enum tamperx);
Function descriptions	disable tamper pin detection
Precondition	-
The called functions	-
Input parameter{in}	
tamperx	refer to Table 3-43. Enum bkp_tamper_enum
TAMPER_0	BKP tamper0
TAMPER_1	BKP tamper1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable tamper0 pin detection */
```

```
bkp_tamper_detection_disable(TAMPER_0);
```

bkp_tamper_active_level_set

The description of bkp_tamper_active_level_set is shown as below:

Table 3-57. Function bkp_tamper_active_level_set

Function name	bkp_tamper_active_level_set
Function prototype	void bkp_tamper_active_level_set(bkp_tamper_enum tamperx, uint16_t level);
Function descriptions	set tamper pin active level
Precondition	-
The called functions	-
Input parameter{in}	
tamperx	refer to Table 3-43. Enum bkp_tamper_enum
TAMPER_0	BKP tamper0
TAMPER_1	BKP tamper1
Input parameter{in}	
level	tamper pin active level
TAMPER_PIN_ACTIVE_HIGH	the tamper pin is active high
TAMPER_PIN_ACTIVE_LOW	the tamper pin is active low

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set tamper0 pin active level high */
```

```
bkp_tamper_active_level_set(TAMPER_0, TAMPER_PIN_ACTIVE_HIGH);
```

bkp_waveform_detect_config

The description of bkp_waveform_detect_config is shown as below:

Table 3-58. Function bkp_waveform_detect_config

Function name	bkp_waveform_detect_config
Function prototype	void bkp_waveform_detect_config (uint16_t waveform_detect_mode, ControlStatus newvalue);
Function descriptions	waveform detect configure
Precondition	-
The called functions	-
Input parameter{in}	
waveform_detect_mode	waveform_detect_mode
BKP_WAVEFORM_DETECT_1	the first waveform detection
BKP_WAVEFORM_DETECT_2	the second waveform detection
Input parameter{in}	
newvalue	control value
ENABLE	enable function
DISABLE	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the first waveform detect */
```

```
bkp_waveform_detect_config(BKP_WAVEFORM_DETECT_1, ENABLE);
```

bkp_flag_get

The description of bkp_flag_get is shown as below:

Table 3-59. Function bkp_flag_get

Function name	bkp_flag_get
Function prototype	FlagStatus bkp_flag_get(uint16_t flag);
Function descriptions	get BKP flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	BKP flag
BKP_FLAG_TAMPER0	tamper0 event flag
BKP_FLAG_TAMPER1 _WAVEDETECT	tamper1/waveform detect event flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get BKP tamper0 event flag */
```

```
FlagStatus status;
```

```
status = bkp_flag_get(BKP_FLAG_TAMPER0);
```

bkp_flag_clear

The description of bkp_flag_clear is shown as below:

Table 3-60. Function bkp_flag_clear

Function name	bkp_flag_clear
Function prototype	void bkp_flag_clear(uint16_t flag);
Function descriptions	clear BKP flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	BKP flag
BKP_FLAG_TAMPER0	tamper0 event flag
BKP_FLAG_TAMPER1 _WAVEDETECT	tamper1/waveform detect event flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear BKP tamper0 event flag */
```

```
bkp_flag_clear(BKP_FLAG_TAMPER0);
```

bkp_tamper_interrupt_enable

The description of bkp_tamper_interrupt_enable is shown as below:

Table 3-61. Function bkp_tamper_interrupt_enable

Function name	bkp_tamper_interrupt_enable
Function prototype	void bkp_tamper_interrupt_enable(uint16_t bkp_interrupt);
Function descriptions	enable tamper interrupt
Precondition	-
The called functions	-
Input parameter{in}	
bkp_interrupt	the BKP interrupt
<i>BKP_INT_TAMPER0</i>	BKP tamper0 interrupt
<i>BKP_INT_TAMPER1_WAVEDETECT</i>	BKP tamper1/waveform detect interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable tamper0 interrupt */
```

```
bkp_tamper_interrupt_enable(BKP_INT_TAMPER0);
```

bkp_tamper_interrupt_disable

The description of bkp_tamper_interrupt_disable is shown as below:

Table 3-62. Function bkp_tamper_interrupt_disable

Function name	bkp_tamper_interrupt_disable
Function prototype	void bkp_tamper_interrupt_disable(uint16_t bkp_interrupt);
Function descriptions	disable tamper interrupt
Precondition	-
The called functions	-
Input parameter{in}	
bkp_interrupt	the BKP interrupt
<i>BKP_INT_TAMPER0</i>	BKP tamper0 interrupt
<i>BKP_INT_TAMPER1_WAVEDETECT</i>	BKP tamper1/waveform detect interrupt
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable tamper0 interrupt */
```

```
bkp_tamper_interrupt_disable(BKP_INT_TAMPER0);
```

bkp_interrupt_flag_get

The description of bkp_interrupt_flag_get is shown as below:

Table 3-63. Function bkp_interrupt_flag_get

Function name	bkp_interrupt_flag_get
Function prototype	FlagStatus bkp_interrupt_flag_get(uint16_t flag);
Function descriptions	get BKP interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	BKP interrupt flag
<i>BKP_INT_FLAG_TAMPER0</i>	tamper0 interrupt flag
<i>BKP_INT_FLAG_TAMPER1_WAVEDETECT</i>	tamper1/waveform detect interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get BKP tamper0 interrupt flag */
```

```
FlagStatus tamper0_status;
```

```
tamper0_status = bkp_interrupt_flag_get(BKP_INT_FLAG_TAMPER0);
```

bkp_interrupt_flag_clear

The description of bkp_interrupt_flag_clear is shown as below:

Table 3-64. Function bkp_interrupt_flag_clear

Function name	bkp_interrupt_flag_clear
Function prototype	void bkp_interrupt_flag_clear(uint16_t flag);
Function descriptions	clear BKP interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	

flag	BKP interrupt flag
<i>BKP_INT_FLAG_TAMP ER0</i>	tamper0 interrupt flag
<i>BKP_INT_FLAG_TAMP ER1_WAVEDETECT</i>	tamper1/waveform detect interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear BKP tamper0 interrupt flag */
```

```
bkp_interrupt_flag_clear(BKP_INT_FLAG_TAMPER0);
```

3.4. 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 [3.4.1](#), the CAN firmware functions are introduced in chapter [3.4.2](#).

3.4.1. Descriptions of Peripheral registers

CAN registers are listed in the table shown as below:

Table 3-65. 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
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

Registers	Descriptions
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_FAFIFO	Filter associated FIFO registe
CAN_FW	Filter working register
CAN_FxDATAy	Filter x data y register

3.4.2. Descriptions of Peripheral functions

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

Table 3-66. CAN firmware function

Function name	Function description
can_deinit	deinitialize CAN
can_struct_para_init	initialize CAN struct
can_init	initialize CAN
can_filter_init	CAN filter initialize
can1_filter_start_bank	set CAN1 fliter 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 trigggle mode enable
can_time_trigger_mode_disable	CAN time trigggle mode disable
can_message_transmit	transmit CAN message
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 3-67. 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_transmit_message_struct

Table 3-68. Structure can_transmit_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 3-69. 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_fi	filtering index

Structure can_filter_parameter_struct

Table 3-70. 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

Enum can_flag_enum

Table 3-71. Enum can_flag_enum

Member name	Function description
CAN_FLAG_RXL	RX level
CAN_FLAG_LASTRX	last sample value of RX pin
CAN_FLAG_RS	receiving state
CAN_FLAG_TS	transmitting state
CAN_FLAG_SLPWF	status change flag of entering sleep working mode
CAN_FLAG_WUWF	status change flag of wakeup from sleep working mode
CAN_FLAG_ERRIF	error flag
CAN_FLAG_SLPWS	sleep working state
CAN_FLAG_IWS	initial working state
CAN_FLAG_TMLS2	transmit mailbox 2 last sending in Tx FIFO
CAN_FLAG_TMLS1	transmit mailbox 1 last sending in Tx FIFO
CAN_FLAG_TMLS0	transmit mailbox 0 last sending in Tx FIFO
CAN_FLAG_TME2	transmit mailbox 2 empty
CAN_FLAG_TME1	transmit mailbox 1 empty
CAN_FLAG_TME0	transmit mailbox 0 empty
CAN_FLAG_MTE2	mailbox 2 transmit error
CAN_FLAG_MTE1	mailbox 1 transmit error
CAN_FLAG_MTE0	mailbox 0 transmit error
CAN_FLAG_MAL2	mailbox 2 arbitration lost
CAN_FLAG_MAL1	mailbox 1 arbitration lost
CAN_FLAG_MAL0	mailbox 0 arbitration lost
CAN_FLAG_MTFNERR2	mailbox 2 transmit finished with no error
CAN_FLAG_MTFNERR1	mailbox 1 transmit finished with no error
CAN_FLAG_MTFNERR0	mailbox 0 transmit finished with no error
CAN_FLAG_MTF2	mailbox 2 transmit finished

Member name	Function description
<i>CAN_FLAG_MTF1</i>	mailbox 1 transmit finished
<i>CAN_FLAG_MTF0</i>	mailbox 0 transmit finished
<i>CAN_FLAG_RFO0</i>	receive FIFO0 overfull
<i>CAN_FLAG_RFF0</i>	receive FIFO0 full
<i>CAN_FLAG_RFO1</i>	receive FIFO1 overfull
<i>CAN_FLAG_RFF1</i>	receive FIFO1 full
<i>CAN_FLAG_BOERR</i>	bus-off error
<i>CAN_FLAG_PERR</i>	passive error
<i>CAN_FLAG_WERR</i>	warning error

Enum `can_interrupt_flag_enum`

Table 3-72. Enum `can_interrupt_flag_enum`

Member name	Function description
<i>CAN_INT_FLAG_SLPIF</i>	status change interrupt flag of sleep working mode entering
<i>CAN_INT_FLAG_WUIF</i>	status change interrupt flag of wakeup from sleep working mode
<i>CAN_INT_FLAG_ERRIF</i>	error interrupt flag
<i>CAN_INT_FLAG_MTF2</i>	mailbox 2 transmit finished interrupt flag
<i>CAN_INT_FLAG_MTF1</i>	mailbox 1 transmit finished interrupt flag
<i>CAN_INT_FLAG_MTF0</i>	mailbox 0 transmit finished interrupt flag
<i>CAN_INT_FLAG_RFO0</i>	receive FIFO0 overfull interrupt flag
<i>CAN_INT_FLAG_RFF0</i>	receive FIFO0 full interrupt flag
<i>CAN_INT_FLAG_RFL0</i>	receive FIFO0 not empty interrupt flag
<i>CAN_INT_FLAG_RFO1</i>	receive FIFO1 overfull interrupt flag
<i>CAN_INT_FLAG_RFF1</i>	receive FIFO1 full interrupt flag
<i>CAN_INT_FLAG_RFL1</i>	receive FIFO1 not empty interrupt flag
<i>CAN_INT_FLAG_ERRN</i>	error number interrupt flag
<i>CAN_INT_FLAG_BOERR</i>	bus-off error interrupt flag
<i>CAN_INT_FLAG_PERR</i>	passive error interrupt flag
<i>CAN_INT_FLAG_WERR</i>	warning error interrupt flag

Enum `can_error_enum`

Table 3-73. Enum `can_error_enum`

Member name	Function description
<i>CAN_ERROR_NONE</i>	no error
<i>CAN_ERROR_FILL</i>	fill error
<i>CAN_ERROR_FORMATE</i>	format error
<i>CAN_ERROR_ACK</i>	ACK error
<i>CAN_ERROR_BITRECESSIVE</i>	bit recessive error
<i>CAN_ERROR_BITDOMINANTER</i>	bit dominant error
<i>CAN_ERROR_CRC</i>	CRC error

Member name	Function description
CAN_ERROR_SOFTWARECFG	software configure

Enum can_transmit_state_enum

Table 3-74. Enum can_transmit_state_enum

Member name	Function description
CAN_TRANSMIT_FAILED	CAN transmitted failure
CAN_TRANSMIT_OK	CAN transmitted success
CAN_TRANSMIT_PENDING	CAN transmitted pending
CAN_TRANSMIT_NOMAILBOX	no empty mailbox to be used for CAN

Enum can_struct_type_enum

Table 3-75. Enum can_struct_type_enum

Member name	Function description
CAN_INIT_STRUCT	CAN initialize 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

can_deinit

The description of can_deinit is shown as below:

Table 3-76. 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	
-	-

Example:

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

can_struct_para_init

The description of can_struct_para_init is shown as below:

Table 3-77. 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	Struct refer to Table 3-75. Enum can_struct_type_enum
CAN_INIT_STRUCT	CAN initialize 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 pointer of the specific struct
Return value	
-	-

Example:

```
/* CAN parameter struct initialize */
can_parameter_struct can_parameter;
can_struct_para_init(CAN_INIT_STRUCT, &can_parameter);
```

can_init

The description of can_init is shown as below:

Table 3-78. 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	-
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 struct, the structure members can refer to members of the structure Table 3-67. Structure can_parameter_struct
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR

Example:

```

/* CAN0 initialize */

can_parameter_struct can_parameter;

can_parameter.time_triggered = DISABLE;

can_parameter.auto_bus_off_recovery = DISABLE;

can_parameter.auto_wake_up = DISABLE;

can_parameter.auto_retrans = ENABLE;

can_parameter.rec_fifo_overwrite = DISABLE;

can_parameter.trans_fifo_order = DISABLE;

can_parameter.working_mode = CAN_NORMAL_MODE;

can_parameter.resync_jump_width = CAN_BT_SJW_1TQ;

can_parameter.time_segment_1 = CAN_BT_BS1_8TQ;

can_parameter.time_segment_2 = CAN_BT_BS2_3TQ;

can_parameter.prescaler = 5;

can_init(CAN0, &can_parameter);

```

can_filter_init

The description of can_filter_init is shown as below:

Table 3-79. 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	-
The called functions	-
Input parameter{in}	
can_filter_parameter_i nit	CAN filter initialization struct, the structure members can refer to members of the structure Table 3-70. Structure can_filter_parameter_struct
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* initialize CAN filter */
```

```
can_filter_parameter_struct can_filter;
```

```
can_filter.filter_number=0;
```

```
can_filter.filter_mode = CAN_FILTERMODE_MASK;
```

```
can_filter.filter_bits = CAN_FILTERBITS_32BIT;
```

```
can_filter.filter_list_high = 0x3000;
```

```
can_filter.filter_list_low = 0x0000;
```

```
can_filter.filter_mask_high = 0x3000;
```

```
can_filter.filter_mask_low = 0x0000;
```

```
can_filter.filter_fifo_number = CAN_FIFO0;
```

```
can_filter.filter_enable = ENABLE;
```

```
can_filter_init(&can_filter);
```

can1_filter_start_bank

The description of can1_filter_start_bank is shown as below:

Table 3-80. 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 fliter 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 3-81. 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
<i>CANx(x=0,1)</i>	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 3-82. 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
<i>CANx(x=0,1)</i>	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 3-83. 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
<i>CANx(x=0,1)</i>	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 3-84. 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
<i>CANx(x=0,1)</i>	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 3-85. 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	-
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 struct, the structure members can refer to members of the structure Table 3-68. Structure can_transmit_message_struct
Output parameter{out}	
-	-
Return value	
uint8_t	0x00-0x03

Example:

```
/* CAN0 transmit message and return the mailbox number*/
uint8_t transmit_mailbox = 0;
can_transmit_message_struct transmit_message;
transmit_message.tx_sfid = 0x300 >> 1;
transmit_message.tx_efid = 0x00;
transmit_message.tx_ft = CAN_FT_DATA;
transmit_message.tx_ff = CAN_FF_STANDARD;
transmit_message.tx_dlen = 2;
transmit_message.tx_data[0] = 0x55;
transmit_message.tx_data[1] = 0xAA;
transmit_mailbox = can_message_transmit(CAN0, &transmit_message);
```

can_transmit_states

The description of can_transmit_states is shown as below:

Table 3-86. 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(x=0,1,2)	CAN_MAILBOXx
Output parameter{out}	
-	-
Return value	
can_transmit_state_enum	Refer to Table 3-74. Enum can_transmit_state_enum

Example:

```
/* CAN0 mailbox0 transmit state */
```

```
can_transmit_state_enum transmit_state = CAN_TRANSMIT_FAILED;
```

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

can_transmission_stop

The description of can_transmission_stop is shown as below:

Table 3-87. Function can_transmission_stop

Function name	can_transmission_stop
Function prototype	ErrStatus 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(x=0,1,2)	CAN_MAILBOXx

Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR

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 3-88. 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	-
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(x=0,1)	CAN_FIFOx
Input parameter{in}	
receive_message	CAN message receive struct, the structure members can refer to members of the structure Table 3-69. Structure can_receive_message_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CAN0 FIFO0 receive message */
can_receive_message_struct receive_message;
can_message_receive(CAN0, CAN_FIFO0, &receive_message);
```

can_fifo_release

The description of can_fifo_release is shown as below:

Table 3-89. Function `can_fifo_release`

Function name	<code>can_fifo_release</code>
Function prototype	<code>void can_fifo_release(uint32_t can_periph, uint8_t fifo_number);</code>
Function descriptions	release FIFO
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(x=0,1)</code>	<code>CAN_FIFOx</code>
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 3-90. Function `can_receive_message_length_get`

Function name	<code>can_receive_message_length_get</code>
Function prototype	<code>uint8_t can_receive_message_length_get(uint32_t can_periph, uint8_t fifo_number);</code>
Function descriptions	CAN receive message length
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(x=0,1)</code>	<code>CAN_FIFOx</code>
Output parameter{out}	
-	-
Return value	
<code>uint8_t</code>	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 3-91. 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 3-92. 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
<i>CANx(x=0,1)</i>	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 3-93. 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
<i>CANx(x=0,1)</i>	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
can_error_enum	Refer to Table 3-73. Enum can_error_enum

Example:

```
/* get CAN0 error type */
can_error_enum err_type;
err_type = can_error_get(CAN0);
```

can_receive_error_number_get

The description of can_receive_error_number_get is shown as below:

Table 3-94. 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
<i>CANx(x=0,1)</i>	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
uint8_t	0..255

Example:

```

/* get CAN0 receive error number */

uint8_t error_num;

error_num = can_receive_error_number_get(CAN0);

```

can_transmit_error_number_get

The description of can_transmit_error_number_get is shown as below:

Table 3-95. 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
<i>CANx(x=0,1)</i>	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
uint8_t	0..255

Example:

```

/* get CAN0 transmit error number */

uint8_t error_num;

error_num = can_transmit_error_number_get(CAN0);

```

can_flag_get

The description of can_flag_get is shown as below:

Table 3-96. Function `can_flag_get`

Function name	<code>can_flag_get</code>
Function prototype	<code>FlagStatus can_flag_get(uint32_t can_periph, can_flag_enum flag);</code>
Function descriptions	get CAN flag state
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<code>CANx(x=0,1)</code>	CAN peripheral selection
Input parameter{in}	
flag	CAN flags refer to Table 3-71. Enum <code>can_flag_enum</code>
<code>CAN_FLAG_RXL</code>	RX level
<code>CAN_FLAG_LASTRX</code>	last sample value of RX pin
<code>CAN_FLAG_RS</code>	receiving state
<code>CAN_FLAG_TS</code>	transmitting state
<code>CAN_FLAG_SLPIF</code>	status change flag of entering sleep working mode
<code>CAN_FLAG_WUIF</code>	status change flag of wakeup from sleep working mode
<code>CAN_FLAG_ERRIF</code>	error flag
<code>CAN_FLAG_SLPWS</code>	sleep working state
<code>CAN_FLAG_IWS</code>	initial working state
<code>CAN_FLAG_TMLS2</code>	transmit mailbox 2 last sending in Tx FIFO
<code>CAN_FLAG_TMLS1</code>	transmit mailbox 1 last sending in Tx FIFO
<code>CAN_FLAG_TMLS0</code>	transmit mailbox 0 last sending in Tx FIFO
<code>CAN_FLAG_TME2</code>	transmit mailbox 2 empty
<code>CAN_FLAG_TME1</code>	transmit mailbox 1 empty
<code>CAN_FLAG_TME0</code>	transmit mailbox 0 empty
<code>CAN_FLAG_MTE2</code>	mailbox 2 transmit error
<code>CAN_FLAG_MTE1</code>	mailbox 1 transmit error
<code>CAN_FLAG_MTE0</code>	mailbox 0 transmit error
<code>CAN_FLAG_MAL2</code>	mailbox 2 arbitration lost
<code>CAN_FLAG_MAL1</code>	mailbox 1 arbitration lost
<code>CAN_FLAG_MAL0</code>	mailbox 0 arbitration lost
<code>CAN_FLAG_MTFNER</code> <code>R2</code>	mailbox 2 transmit finished with no error
<code>CAN_FLAG_MTFNER</code> <code>R1</code>	mailbox 1 transmit finished with no error
<code>CAN_FLAG_MTFNER</code> <code>R0</code>	mailbox 0 transmit finished with no error
<code>CAN_FLAG_MTF2</code>	mailbox 2 transmit finished
<code>CAN_FLAG_MTF1</code>	mailbox 1 transmit finished
<code>CAN_FLAG_MTF0</code>	mailbox 0 transmit finished
<code>CAN_FLAG_RFO0</code>	receive FIFO0 overfull

<code>CAN_FLAG_RFF0</code>	receive FIFO0 full
<code>CAN_FLAG_RFO1</code>	receive FIFO1 overfull
<code>CAN_FLAG_RFF1</code>	receive FIFO1 full
<code>CAN_FLAG_BOERR</code>	bus-off error
<code>CAN_FLAG_PERR</code>	passive error
<code>CAN_FLAG_WERR</code>	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 3-97. 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
<code>CANx(x=0,1)</code>	CAN peripheral selection
Input parameter{in}	
flag	CAN flags refer to Table 3-71. Enum can_flag_enum
<code>CAN_FLAG_SLPIF</code>	status change flag of entering sleep working mode
<code>CAN_FLAG_WUIF</code>	status change flag of wakeup from sleep working mode
<code>CAN_FLAG_ERRIF</code>	error flag
<code>CAN_FLAG_MTE2</code>	mailbox 2 transmit error
<code>CAN_FLAG_MTE1</code>	mailbox 1 transmit error
<code>CAN_FLAG_MTE0</code>	mailbox 0 transmit error
<code>CAN_FLAG_MAL2</code>	mailbox 2 arbitration lost
<code>CAN_FLAG_MAL1</code>	mailbox 1 arbitration lost
<code>CAN_FLAG_MAL0</code>	mailbox 0 arbitration lost
<code>CAN_FLAG_MTFNER</code> <code>R2</code>	mailbox 2 transmit finished with no error
<code>CAN_FLAG_MTFNER</code> <code>R1</code>	mailbox 1 transmit finished with no error

<i>CAN_FLAG_MTFNER</i> <i>R0</i>	mailbox 0 transmit finished with no error
<i>CAN_FLAG_MTF2</i>	mailbox 2 transmit finished
<i>CAN_FLAG_MTF1</i>	mailbox 1 transmit finished
<i>CAN_FLAG_MTF0</i>	mailbox 0 transmit finished
<i>CAN_FLAG_RFO0</i>	receive FIFO0 overfull
<i>CAN_FLAG_RFF0</i>	receive FIFO0 full
<i>CAN_FLAG_RFO1</i>	receive FIFO1 overfull
<i>CAN_FLAG_RFF1</i>	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 3-98. 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
<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 enable */
```

```
can_interrupt_enable(CAN0, CAN_INT_TME);
```

can_interrupt_disable

The description of can_interrupt_disable is shown as below:

Table 3-99. 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}	
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
<i>CAN_INT_RFNE0</i>	receive FIFO0 not empty interrupt
<i>CAN_INT_RFF0</i>	receive FIFO0 full interrupt
<i>CAN_INT_RFO0</i>	receive FIFO0 overfull interrupt
<i>CAN_INT_RFNE1</i>	receive FIFO1 not empty interrupt
<i>CAN_INT_RFF1</i>	receive FIFO1 full interrupt
<i>CAN_INT_RFO1</i>	receive FIFO1 overfull interrupt
<i>CAN_INT_WERR</i>	warning error interrupt
<i>CAN_INT_PERR</i>	passive error interrupt
<i>CAN_INT_BO</i>	bus-off interrupt
<i>CAN_INT_ERRN</i>	error number interrupt
<i>CAN_INT_ERR</i>	error interrupt
<i>CAN_INT_WU</i>	wakeup interrupt
<i>CAN_INT_SLPW</i>	sleep working interrupt
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 3-100. Function can_interrupt_flag_get

Function name	can_interrupt_flag_get
Function prototype	FlagStatus can_interrupt_flag_get(uint32_t can_periph, can_interrupt_flag_enum flag);
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 refer to Table 3-72. Enum can_interrupt_flag_enum
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_RFO0	receive FIFO0 overfull interrupt flag
CAN_INT_FLAG_RFF0	receive FIFO0 full interrupt flag
CAN_INT_FLAG_RFO1	receive FIFO1 overfull interrupt flag
CAN_INT_FLAG_RFF1	receive FIFO1 full interrupt flag
CAN_INT_FLAG_ERR N	error number interrupt flag
CAN_INT_FLAG_BOE RR	bus-off error interrupt flag
CAN_INT_FLAG_PER R	passive error interrupt flag
CAN_INT_FLAG_WER	warning error interrupt flag

<i>R</i>	
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* get CAN0 mailbox 0 transmit finished interrupt flag */
```

```
FlagStatus Status;
```

```
Status = 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 3-101. 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}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Input parameter{in}	
flag	CAN interrupt flags refer to Table 3-72. Enum can_interrupt_flag_enum
<i>CAN_INT_FLAG_SLPIF</i>	status change interrupt flag of sleep working mode entering
<i>CAN_INT_FLAG_WUIF</i>	status change interrupt flag of wakeup from sleep working mode
<i>CAN_INT_FLAG_ERRIF</i>	error interrupt flag
<i>CAN_INT_FLAG_MTF2</i>	mailbox 2 transmit finished interrupt flag
<i>CAN_INT_FLAG_MTF1</i>	mailbox 1 transmit finished interrupt flag
<i>CAN_INT_FLAG_MTF0</i>	mailbox 0 transmit finished interrupt flag
<i>CAN_INT_FLAG_RFO0</i>	receive FIFO0 overfull interrupt flag
<i>CAN_INT_FLAG_RFF0</i>	receive FIFO0 full interrupt flag
<i>CAN_INT_FLAG_RFO1</i>	receive FIFO1 overfull interrupt flag
<i>CAN_INT_FLAG_RFF1</i>	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);
```

3.5. CAU

The Cryptographic Acceleration Unit supports acceleration of DES, Triple-DES or AES (128,192, or 256) algorithms. The CAU registers are listed in chapter [3.5.1](#) the CAU firmware functions are introduced in chapter [3.5.2](#)

3.5.1. Descriptions of Peripheral registers

CAU registers are listed in the table shown as below:

Table 3-102. CAU Registers

Registers	Descriptions
CAU_CTL	control register
CAU_STAT0	status register 0
CAU_DI	data input register
CAU_DO	data output register
CAU_DMAEN	DMA enable register
CAU_INTEN	interrupt enable register
CAU_STAT1	status register 1
CAU_INTF	interrupt flag register
CAU_KEY0H	key 0 high register
CAU_KEY0L	key 0 low register
CAU_KEY1H	key 1 high register
CAU_KEY1L	key 1 low register
CAU_KEY2H	key 2 high register
CAU_KEY2L	key 2 low register
CAU_KEY3H	key 3 high register
CAU_KEY3L	key 3 low register
CAU_IV0H	initial vector 0 high register
CAU_IV0L	initial vector 0 low register
CAU_IV1H	initial vector 1 high register
CAU_IV1L	initial vector 1 low register

3.5.2. Descriptions of Peripheral functions

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

Table 3-103. CAU firmware function

Function name	Function description
cau_deinit	reset the CAU peripheral
cau_enable	enable the CAU peripheral
cau_disable	disable the CAU peripheral
cau_dma_enable	enable the CAU DMA interface
cau_dma_disable	disable the CAU DMA interface
cau_init	initialize the CAU peripheral
cau_aes_keysize_config	configure key size if used AES algorithm
cau_key_init	initialize the key parameters
cau_key_parameter_init	initialize the struct cau_key_initpara
cau_iv_init	initialize the vectors parameters
cau_iv_parameter_init	initialize the struct cau_iv_parameter_struct
cau_fifo_flush	flush the IN and OUT FIFOs
cau_enable_state_get	return whether CAU peripheral is enabled or disabled
cau_data_write	write data to the IN FIFO
cau_data_read	return the last data entered into the output FIFO
cau_aes_ecb	encrypt and decrypt using AES in ECB mode
cau_aes_cbc	encrypt and decrypt using AES in CBC mode
cau_aes_ctr	encrypt and decrypt using AES in CTR mode
cau_tdes_ecb	encrypt and decrypt using TDES in ECB mode
cau_tdes_cbc	encrypt and decrypt using TDES in CBC mode
cau_des_ecb	encrypt and decrypt using DES in ECB mode
cau_des_cbc	encrypt and decrypt using DES in CBC mode
cau_flag_get	get the CAU flag status
cau_interrupt_enable	enable the CAU interrupts
cau_interrupt_disable	disable the CAU interrupts
cau_interrupt_flag_get	get the interrupt flag

Structure cau_key_parameter_struct

Table 3-104. Structure cau_key_parameter_struct

Member name	Function description
key_0_high	key 0 high
key_0_low	key 0 low
key_1_high	key 1 high
key_1_low	key 1 low
key_2_high	key 2 high
key_2_low	key 2 low
key_3_high	key 3 high
key_3_low	key 3 low

Structure cau_iv_parameter_struct

Table 3-105. Structure cau_iv_parameter_struct

Member name	Function description
iv_0_high	init vector 0 high
iv_0_low	init vector 0 low
iv_1_high	init vector 1 high
iv_1_low	init vector 1 low

Enum cau_text_struct

Table 3-106. Enum cau_text_struct

Member name	Function description
input	pointer to the input buffer
in_length	length of the input buffer, must be a multiple of 8(DES and TDES) or 16(AES)
output	pointer to the returned buffer

cau_deinit

The description of cau_deinit is shown as below:

Table 3-107. Function cau_deinit

Function name	cau_deinit
Function prototype	void cau_deinit(void);
Function descriptions	reset the CAU peripheral
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the CAU peripheral */
cau_deinit();
```

cau_enable

The description of cau_enable is shown as below:

Table 3-108. Function cau_enable

Function name	cau_enable
Function prototype	void cau_enable(void);
Function descriptions	enable the CAU peripheral
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CAU peripheral */
cau_enable();
```

cau_disable

The description of cau_disable is shown as below:

Table 3-109. Function cau_disable

Function name	cau_disable
Function prototype	void cau_disable(void);
Function descriptions	disable the CAU peripheral
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CAU peripheral */
cau_disable();
```

cau_dma_enable

The description of cau_dma_enable is shown as below:

Table 3-110. Function cau_dma_enable

Function name	cau_dma_enable
----------------------	----------------

Function prototype	void cau_dma_enable(uint32_t dma_req)
Function descriptions	enable the CAU DMA interface
Precondition	-
The called functions	-
Input parameter{in}	
dma_req	specify the CAU DMA transfer request to be enabled
CAU_DMA_INFIFO	DMA for incoming(Rx) data transfer
CAU_DMA_OUTFIFO	DMA for outgoing(Tx) data transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CAU DMA interface */
```

```
cau_dma_enable(CAU_DMA_INFIFO);
```

cau_dma_disable

The description of cau_dma_disable is shown as below:

Table 3-111. Function cau_dma_disable

Function name	cau_dma_disable
Function prototype	void cau_dma_disable(uint32_t dma_req);
Function descriptions	disable the CAU DMA interface
Precondition	-
The called functions	-
Input parameter{in}	
dma_req	specify the CAU DMA transfer request to be disabled
CAU_DMA_INFIFO	DMA for incoming(Rx) data transfer
CAU_DMA_OUTFIFO	DMA for outgoing(Tx) data transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CAU DMA interface */
```

```
cau_dma_disable(CAU_DMA_INFIFO);
```

cau_init

The description of cau_init is shown as below:

Table 3-112. Function cau_init

Function name	cau_init
Function prototype	void cau_init(uint32_t algo_dir, uint32_t algo_mode, uint32_t swapping);
Function descriptions	initialize the CAU peripheral
Precondition	-
The called functions	-
Input parameter{in}	
algo_dir	algorithm direction
CAU_ENCRYPT	encrypt
CAU_DECRYPT	decrypt
Input parameter{in}	
algo_mode	algorithm mode selection
CAU_MODE_TDES_ECB	TDES-ECB (3DES Electronic codebook)
CAU_MODE_TDES_CBC	TDES-CBC (3DES Cipher block chaining)
CAU_MODE_DES_ECB	DES-ECB (simple DES Electronic codebook)
CAU_MODE_DES_CBC	DES-CBC (simple DES Cipher block chaining)
CAU_MODE_AES_ECB	AES-ECB (AES Electronic codebook)
CAU_MODE_AES_CBC	AES-CBC (AES Cipher block chaining)
CAU_MODE_AES_CTR	AES-CTR (AES counter mode)
CAU_MODE_AES_KEY	AES decryption key preparation mode
Input parameter{in}	
swapping	data swapping selection
CAU_SWAPPING_32BIT	no swapping
CAU_SWAPPING_16BIT	half-word swapping
CAU_SWAPPING_8BIT	bytes swapping
CAU_SWAPPING_1BIT	bit swapping
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the CAU peripheral */
```

```
cau_init(CAU_ENCRYPT, CAU_MODE_TDES_ECB, CAU_SWAPPING_32BIT);
```

cau_aes_keysize_config

The description of cau_aes_keysize_config is shown as below:

Table 3-113. Function cau_aes_keysize_config

Function name	cau_aes_keysize_config
Function prototype	void cau_aes_keysize_config(uint32_t key_size);
Function descriptions	configure key size if used AES algorithm
Precondition	-
The called functions	-
Input parameter{in}	
key_size	key length selection when aes mode
CAU_KEYSIZE_128BIT	128 bit key length
CAU_KEYSIZE_192BIT	192 bit key length
CAU_KEYSIZE_256BIT	256 bit key length
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure key size if used AES algorithm */
```

```
cau_aes_keysize_config(CAU_KEYSIZE_128BIT);
```

cau_key_init

The description of cau_key_init is shown as below:

Table 3-114. Function cau_key_init

Function name	cau_key_init
Function prototype	void cau_key_init(cau_key_parameter_struct* key_initpara);
Function descriptions	initialize the key parameters
Precondition	-
The called functions	-
Input parameter{in}	
key_initpara	the key parameters, refer to structure Table 3-104. Structure cau_key_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the key parameters */
```

```
cau_key_parameter_struct key_initpara;
```

```
uint32_t key_128[4] = {0x2b7e1516U, 0x28aed2a6U, 0xabf71588U, 0x09cf4f3cU};
```

```
key_initpara.key_2_high = key_128[0];
```

```
key_initpara.key_2_low  = key_128[1];
```

```
key_initpara.key_3_high = key_128[2];
```

```
key_initpara.key_3_low  = key_128[3];
```

```
cau_key_init(&key_initpara);
```

cau_key_parameter_init

The description of cau_key_parameter_init is shown as below:

Table 3-115. Function cau_key_parameter_init

Function name	cau_key_parameter_init
Function prototype	void cau_key_parameter_init(cau_key_parameter_struct* key_initpara);
Function descriptions	initialize the sturct cau_key_initpara
Precondition	-
The called functions	-
Input parameter{in}	
key_initpara	the key parameters, refer to structure Table 3-104. Structure cau_key_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the sturct cau_key_initpara */
```

```
cau_key_parameter_struct key_initpara;
```

```
cau_key_parameter_init(&key_initpara);
```

cau_iv_init

The description of cau_iv_init is shown as below:

Table 3-116. Function cau_iv_init

Function name	cau_iv_init
Function prototype	void cau_iv_init(cau_iv_parameter_struct* iv_initpara);
Function descriptions	initialize the vectors parameters

Precondition	-
The called functions	-
Input parameter{in}	
iv_initpara	the vectors parameter, refer to structure Table 3-105. Structure <u>cau_iv_parameter_struct</u>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize the vectors parameters */

cau_iv_parameter_struct iv_initpara;

uint32_t vectors[4] = {0x00010203U, 0x04050607U, 0x08090A0BU, 0x0C0D0E0FU};

iv_initpara.iv_0_high = vectors[0];
iv_initpara.iv_0_low = vectors[1];
iv_initpara.iv_1_high = vectors[2];
iv_initpara.iv_1_low = vectors[3];

cau_iv_init(&iv_initpara);

```

cau_iv_parameter_init

The description of cau_iv_parameter_init is shown as below:

Table 3-117. Function cau_iv_parameter_init

Function name	cau_iv_parameter_init
Function prototype	void cau_iv_parameter_init(cau_iv_parameter_struct* iv_initpara);
Function descriptions	initialize the vectors parameters
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
iv_initpara	the vectors parameter, refer to structure Table 3-105. Structure <u>cau_iv_parameter_struct</u>
Return value	
-	-

Example:

```

/* initialize the vectors parameters */

```

```
cau_iv_parameter_struct iv_initpara;
```

```
cau_iv_parameter_init(&iv_initpara);
```

cau_fifo_flush

The description of cau_fifo_flush is shown as below:

Table 3-118. Function cau_fifo_flush

Function name	cau_fifo_flush
Function prototype	void cau_fifo_flush(void);
Function descriptions	flush the IN and OUT FIFOs
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* flush the IN and OUT FIFOs */
cau_fifo_flush();
```

cau_enable_state_get

The description of cau_enable_state_get is shown as below:

Table 3-119. Function cau_enable_state_get

Function name	cau_enable_state_get
Function prototype	ControlStatus cau_enable_state_get(void);
Function descriptions	return whether CAU peripheral is enabled or disabled
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ControlStatus	ENABLE or DISABLE

Example:

```
/* return whether CAU peripheral is enabled or disabled */
```



```
ControlStatus state;
```

```
state = cau_enable_state_get();
```

cau_data_write

The description of cau_data_write is shown as below:

Table 3-120. Function cau_data_write

Function name	cau_data_write
Function prototype	void cau_data_write(uint32_t data);
Function descriptions	write data to the IN FIFO
Precondition	-
The called functions	-
Input parameter{in}	
data	data to write(0x0 - 0xFFFFFFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write data to the IN FIFO */
cau_data_write(0x10);
```

cau_data_read

The description of cau_data_read is shown as below:

Table 3-121. Function cau_data_read

Function name	cau_data_read
Function prototype	uint32_t cau_data_read(void);
Function descriptions	return the last data entered into the output FIFO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	0x0 – 0xFFFFFFFF

Example:

```
/* return the last data entered into the output FIFO */
```

```
uint32_t data;

data = cau_data_read();
```

cau_aes_ecb

The description of cau_aes_ecb is shown as below:

Table 3-122. Function cau_aes_ecb

Function name	cau_aes_ecb
Function prototype	ErrStatus cau_aes_ecb(uint32_t algo_dir, uint8_t *key, uint16_t keysize, cau_text_struct *text);
Function descriptions	encrypt and decrypt using AES in ECB mode
Precondition	-
The called functions	-
Input parameter{in}	
algo_dir	algorithm direction
CAU_ENCRYPT	encrypt
CAU_DECRYPT	decrypt
Input parameter{in}	
key	key used for AES algorithm
Input parameter{in}	
keysize	length of the key, must be a 128, 192 or 256
Input parameter{in}	
text	the text parameter, refer to structure Table 3-106. Enum cau_text_struct
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* encrypt using AES in ECB mode */

ErrStatus status;

cau_text_struct text;

uint8_t plaintext[64] = {

    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,

    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,

    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,

    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,

    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
```

```

0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
};

uint8_t encrypt_result[64];

text.input = plaintext;

text.in_length = 64;

text.output = encrypt_result;

status = cau_aes_ecb(CAU_ENCRYPT, CAU_KEYSIZE_128BIT, 128, &text);

```

cau_aes_cbc

The description of cau_aes_cbc is shown as below:

Table 3-123. Function cau_aes_cbc

Function name	cau_aes_cbc
Function prototype	ErrStatus cau_aes_cbc(uint32_t algo_dir, uint8_t *key, uint16_t keysize, uint8_t iv[16], cau_text_struct *text);
Function descriptions	encrypt and decrypt using AES in CBC mode
Precondition	-
The called functions	-
Input parameter{in}	
algo_dir	algorithm direction
CAU_ENCRYPT	encrypt
CAU_DECRYPT	decrypt
Input parameter{in}	
key	key used for AES algorithm
Input parameter{in}	
keysize	length of the key, must be a 128, 192 or 256
Input parameter{in}	
iv	initialization vectors used for AES algorithm
Input parameter{in}	
text	the text parameter, refer to structure Table 3-106. Enum cau_text_struct
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* encrypt using AES in CBC mode */
```

```

ErrStatus status;

cau_text_struct text;

uint8_t plaintext[64] = {

    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,

    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,

    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,

    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,

    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,

    0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,

    0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,

    0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U

};

uint8_t encrypt_result[64];

text.input = plaintext;

text.in_length = 64;

text.output = encrypt_result;

uint8_t iv[16] = {0x00}

status = cau_aes_cbc(CAU_ENCRYPT ,CAU_KEYSIZE_128BIT, 128, iv, &text);

```

cau_aes_ctr

The description of cau_aes_ctr is shown as below:

Table 3-124. Function cau_aes_ctr

Function name	cau_aes_ctr
Function prototype	ErrStatus cau_aes_ctr(uint32_t algo_dir, uint8_t *key, uint16_t keysize, uint8_t iv[16], cau_text_struct *text);
Function descriptions	encrypt and decrypt using AES in CTR mode
Precondition	-
The called functions	-
Input parameter{in}	
algo_dir	algorithm direction
CAU_ENCRYPT	encrypt
CAU_DECRYPT	decrypt
Input parameter{in}	
key	key used for AES algorithm

Input parameter{in}	
keysize	length of the key, must be a 128, 192 or 256
Input parameter{in}	
iv	initialization vectors used for AES algorithm
Input parameter{in}	
text	the text parameter, refer to structure Table 3-106. Enum cau_text_struct
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* encrypt using AES in CTR mode */
```

```
ErrStatus status;
```

```
cau_text_struct text;
```

```
uint8_t plaintext[64] = {
```

```
    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
    0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
    0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
    0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
```

```
};
```

```
uint8_t encrypt_result[64];
```

```
text.input = plaintext;
```

```
text.in_length = 64;
```

```
text.output = encrypt_result;
```

```
uint8_t iv[16] = {0x00}
```

```
status = cau_aes_ctr(CAU_ENCRYPT, CAU_KEYSZIE_128BIT, 128, iv, &text);
```

cau_tdes_ecb

The description of cau_tdes_ecb is shown as below:

Table 3-125. Function cau_tdes_ecb

Function name	cau_tdes_ecb
Function prototype	ErrStatus cau_tdes_ecb(uint32_t algo_dir, uint8_t key[24], cau_text_struct *text);
Function descriptions	encrypt and decrypt using TDES in ECB mode
Precondition	-
The called functions	-
Input parameter{in}	
algo_dir	algorithm direction
CAU_ENCRYPT	encrypt
CAU_DECRYPT	decrypt
Input parameter{in}	
key	key used for TDES algorithm
Input parameter{in}	
text	the text parameter, refer to structure Table 3-106. Enum cau_text_struct
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* encrypt using TDES in ECB mode */
```

```
ErrStatus status;
```

```
cau_text_struct text;
```

```
uint8_t plaintext[64] = {
```

```
    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
    0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
    0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
    0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
```

```
};
```

```
uint8_t encrypt_result[64];
```

```
text.input = plaintext;
```

```
text.in_length = 64;
```

```
text.output = encrypt_result;
```

```
uint8_t key[24] = {0x01};
```

```
status = cau_tdes_ecb(CAU_ENCRYPT, key, &text);
```

cau_tdes_cbc

The description of cau_tdes_cbc is shown as below:

Table 3-126. Function cau_tdes_cbc

Function name	cau_tdes_cbc
Function prototype	ErrStatus cau_tdes_cbc(uint32_t algo_dir, uint8_t key[24], uint8_t iv[8], cau_text_struct *text);
Function descriptions	encrypt and decrypt using TDES in CBC mode
Precondition	-
The called functions	-
Input parameter{in}	
algo_dir	algorithm direction
CAU_ENCRYPT	encrypt
CAU_DECRYPT	decrypt
Input parameter{in}	
key	key used for TDES algorithm
Input parameter{in}	
iv	initialization vectors used for TDES algorithm
Input parameter{in}	
text	the text parameter, refer to structure Table 3-106. Enum cau_text_struct
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* encrypt using TDES in CBC mode */
```

```
ErrStatus status;
```

```
cau_text_struct text;
```

```
uint8_t plaintext[64] = {
```

```
    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
```

```
    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
```

```
    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
```

```
0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
};
```

```
uint8_t encrypt_result[64];

text.input = plaintext;

text.in_length = 64;

text.output = encrypt_result;

uint8_t key[24] = {0x01};

uint8_t iv[8] = {0x00};

status = cau_tdes_cbc(CAU_ENCRYPT, key, iv, &text);
```

cau_des_ecb

The description of cau_des_ecb is shown as below:

Table 3-127. Function cau_des_ecb

Function name	cau_des_ecb
Function prototype	ErrStatus cau_des_ecb(uint32_t algo_dir, uint8_t *key, uint16_t keysize, cau_text_struct *text);
Function descriptions	encrypt and decrypt using DES in ECB mode
Precondition	-
The called functions	-
Input parameter{in}	
algo_dir	algorithm direction
CAU_ENCRYPT	encrypt
CAU_DECRYPT	decrypt
Input parameter{in}	
key	key used for DES algorithm
Input parameter{in}	
keysize	length of the key, must be a 128, 192 or 256
Input parameter{in}	
text	the text parameter, refer to structure Table 3-106. Enum cau_text_struct
Output parameter{out}	
-	-
Return value	

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

Example:

```
/* encrypt using DES in ECB mode */
```

```
ErrStatus status;
```

```
cau_text_struct text;
```

```
uint8_t plaintext[64] = {
```

```
    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
    0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
    0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
    0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
```

```
};
```

```
uint8_t encrypt_result[64];
```

```
text.input = plaintext;
```

```
text.in_length = 64;
```

```
text.output = encrypt_result;
```

```
uint8_t key[24] = {0x01};
```

```
status = cau_des_ecb(CAU_ENCRYPT, key, &text);
```

cau_des_cbc

The description of cau_des_cbc is shown as below:

Table 3-128. Function cau_des_cbc

Function name	cau_des_cbc
Function prototype	ErrStatus cau_des_cbc(uint32_t algo_dir, uint8_t key[24], uint8_t iv[8], cau_text_struct *text);
Function descriptions	encrypt and decrypt using DES in CBC mode
Precondition	-
The called functions	-
Input parameter{in}	

algo_dir	algorithm direction
<i>CAU_ENCRYPT</i>	encrypt
<i>CAU_DECRYPT</i>	decrypt
Input parameter{in}	
key	key used for DES algorithm
Input parameter{in}	
iv	initialization vectors used for DES algorithm
Input parameter{in}	
text	the text parameter, refer to structure Table 3-106. Enum cau_text_struct
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* encrypt using DES in CBC mode */
```

```
ErrStatus status;
```

```
cau_text_struct text;
```

```
uint8_t plaintext[64] = {
```

```
    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
    0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
    0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
    0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
```

```
};
```

```
uint8_t encrypt_result[64];
```

```
text.input = plaintext;
```

```
text.in_length = 64;
```

```
text.output = encrypt_result;
```

```
uint8_t key[24] = {0x01};
```

```
uint8_t iv[8] = {0x00};
```

```
status = cau_des_cbc(CAU_ENCRYPT, key, iv, &text);
```

cau_flag_get

The description of cau_flag_get is shown as below:

Table 3-129. Function cau_flag_get

Function name	cau_flag_get
Function prototype	FlagStatus cau_flag_get(uint32_t flag);
Function descriptions	get the CAU flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	CAU flag status
CAU_FLAG_INFIFO_EMPTY	input FIFO empty
CAU_FLAG_INFIFO_NO_FULL:	input FIFO is not full
CAU_FLAG_OUTFIFO_NO_EMPTY	output FIFO not empty
CAU_FLAG_OUTFIFO_FULL	output FIFO is full
CAU_FLAG_BUSY	the CAU core is busy
CAU_FLAG_INFIFO	input FIFO flag status
CAU_FLAG_OUTFIFO	output FIFO flag status
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the CAU flag status */
```

```
FlagStatus status;
```

```
status = cau_flag_get(CAU_FLAG_INFIFO_EMPTY);
```

cau_interrupt_enable

The description of cau_interrupt_enable is shown as below:

Table 3-130. Function cau_interrupt_enable

Function name	cau_interrupt_enable
Function prototype	void cau_interrupt_enable(uint32_t interrupt);
Function descriptions	enable the CAU interrupts
Precondition	-

The called functions	-
Input parameter{in}	
interrupt	specify the CAU interrupt source to be enabled
<i>CAU_INT_INFIFO</i>	input FIFO interrupt
<i>CAU_INT_OUTFIFO</i>	output FIFO interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable CAU interrupt */
cau_interrupt_enable(CAU_INT_INFIFO);
```

cau_interrupt_disable

The description of cau_interrupt_disable is shown as below:

Table 3-131. Function cau_interrupt_disable

Function name	cau_interrupt_disable
Function prototype	void cau_interrupt_disable(uint32_t interrupt);
Function descriptions	disable the CAU interrupts
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify the CAU interrupt source to be disabled
<i>CAU_INT_INFIFO</i>	input FIFO interrupt
<i>CAU_INT_OUTFIFO</i>	output FIFO interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable CAU interrupt */
cau_interrupt_disable(CAU_INT_INFIFO);
```

cau_interrupt_flag_get

The description of cau_interrupt_flag_get is shown as below:

Table 3-132. Function cau_interrupt_flag_get

Function name	cau_interrupt_flag_get
Function prototype	FlagStatus cau_interrupt_flag_get(uint32_t int_flag);

Function descriptions	get the interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	CAU interrupt flag
CAU_INT_FLAG_INFIFO	input FIFO interrupt
CAU_INT_FLAG_OUTFIFO	output FIFO interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```

/* get the CAU interrupt flag status */

FlagStatus status;

status = cau_interrupt_flag_get(CAU_INT_FLAG_INFIFO);

```

3.6. 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 [3.6.1](#), the CRC firmware functions are introduced in chapter [3.6.2](#).

3.6.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

Table 3-133. CRC Registers

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

3.6.2. Descriptions of Peripheral functions

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

Table 3-134. CRC firmware function

Function name	Function description
crc_deinit	deinit CRC calculation unit

Function name	Function description
crc_data_register_reset	reset data register to the value of initializaiton data register
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 3-135. 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 3-136. Function crc_data_register_reset

Function name	crc_data_register_reset
Function prototype	void crc_data_register_reset(void);
Function descriptions	reset data register to the value of initializaiton data register
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 3-137. 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	
<code>uint32_t</code>	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 3-138. Function `crc_free_data_register_read`

Function name	<code>crc_free_data_register_read</code>
Function prototype	<code>uint8_t crc_free_data_register_read(void);</code>
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 3-139. Function `crc_free_data_register_write`

Function name	<code>crc_free_data_register_write</code>
Function prototype	<code>void crc_free_data_register_write(uint8_t free_data);</code>
Function descriptions	write data to the free data register
Precondition	-
The called functions	-
Input parameter{in}	
free_data	specify 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 3-140. Function `crc_single_data_calculate`

Function name	<code>crc_single_data_calculate</code>
Function prototype	<code>uint32_t crc_single_data_calculate(uint32_t sdata);</code>
Function descriptions	calculate the CRC value of a 32-bit data
Precondition	-
The called functions	-
Input parameter{in}	
sdata	specify 32-bit data
Output parameter{out}	
-	-
Return value	

uint32_t	32-bit CRC calculate value (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 3-141. 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 data words
Input parameter{in}	
size	size of the array
Output parameter{out}	
-	-
Return value	
uint32_t	32-bit CRC calculate value (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);
```

3.7. 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 [3.7.1](#) the DAC firmware functions are introduced in chapter [3.7.2](#).

3.7.1. Peripheral register description

DAC registers are listed in the table shown as below:

Table 3-142. DAC Registers

Register	Descriptions
DAC_CTL0	DACx control register 0
DAC_SWT	DACx software trigger register
DAC_OUT0_R12DH	DACx_OUT0 12-bit right-aligned data holding register
DAC_OUT0_L12DH	DACx_OUT0 12-bit left-aligned data holding register
DAC_OUT0_R8DH	DACx_OUT0 8-bit right-aligned data holding register
DAC_OUT1_R12DH	DACx_OUT1 12-bit right-aligned data holding register
DAC_OUT1_L12DH	DACx_OUT1 12-bit left-aligned data holding register
DAC_OUT1_R8DH	DACx_OUT1 8-bit right-aligned data holding register
DACC_R12DH	DACx concurrent mode 12-bit right-aligned data holding register
DACC_L12DH	DACx concurrent mode 12-bit left-aligned data holding register
DACC_R8DH	DACx concurrent mode 8-bit right-aligned data holding register
DAC_OUT0_DO	DACx_OUT0 data output register
DAC_OUT1_DO	DACx_OUT1 data output register

3.7.2. Descriptions of Peripheral functions

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

Table 3-143. DAC firmware functions

Function name	Function description
dac_deinit	deinitialize DAC
dac_enable	enable DAC
dac_disable	disable DAC
dac_dma_enable	enable DAC DMA function
dac_dma_disable	disable DAC DMA function
dac_output_buffer_enable	enable DAC output buffer
dac_output_buffer_disable	disable DAC output buffer
dac_output_value_get	get DAC output value
dac_data_set	set DAC data holding register value
dac_trigger_enable	enable DAC trigger
dac_trigger_disable	disable DAC trigger

Function name	Function description
<code>dac_trigger_source_config</code>	configure DAC trigger source
<code>dac_software_trigger_enable</code>	enable DAC software trigger
<code>dac_wave_mode_config</code>	configure DAC wave mode
<code>dac_lfsr_noise_config</code>	configure DAC LFSR noise mode
<code>dac_triangle_noise_config</code>	configure DAC triangle noise mode
<code>dac_concurrent_enable</code>	enable DAC concurrent mode
<code>dac_concurrent_disable</code>	disable DAC concurrent mode
<code>dac_concurrent_software_trigger_enable</code>	enable DAC concurrent software trigger
<code>dac_concurrent_output_buffer_enable</code>	enable DAC concurrent buffer function
<code>dac_concurrent_output_buffer_disable</code>	disable DAC concurrent buffer function
<code>dac_concurrent_data_set</code>	set DAC concurrent mode data holding register value

dac_deinit

The description of `dac_deinit` is shown as below:

Table 3-144. Function `dac_deinit`

Function name	<code>dac_deinit</code>
Function prototype	<code>void dac_deinit(uint32_t dac_periph);</code>
Function descriptions	deinitialize DAC
Precondition	-
The called functions	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
Input parameter{in}	
<code>dac_periph</code>	DAC peripheral
<code>DACx</code>	DAC peripheral selection(x = 0)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize DAC0 */
```

```
dac_deinit(DAC0);
```

dac_enable

The description of `dac_enable` is shown as below:

Table 3-145. Function `dac_enable`

Function name	<code>dac_enable</code>
Function prototype	<code>void dac_enable(uint32_t dac_periph, uint8_t dac_out);</code>
Function descriptions	enable DAC

Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC0_OUT0 */
dac_enable(DAC0, DAC_OUT0);
```

dac_disable

The description of dac_disable is shown as below:

Table 3-146. Function dac_disable

Function name	dac_disable
Function prototype	void dac_disable(uint32_t dac_periph, uint8_t dac_out);
Function descriptions	disable DAC
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC0_OUT0 */
dac_disable(DAC0, DAC_OUT0);
```

dac_dma_enable

The description of dac_dma_enable is shown as below:

Table 3-147. Function dac_dma_enable

Function name	dac_dma_enable
Function prototype	void dac_dma_enable(uint32_t dac_periph, uint8_t dac_out);
Function descriptions	enable DAC DMA function
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC0_OUT0 DMA function */
dac_dma_enable(DAC0, DAC_OUT0);
```

dac_dma_disable

The description of dac_dma_disable is shown as below:

Table 3-148. Function dac_dma_disable

Function name	dac_dma_disable
Function prototype	void dac_dma_disable(uint32_t dac_periph, uint8_t dac_out);
Function descriptions	disable DAC DMA function
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable DAC0_OUT0 DMA function */
```

```
dac_dma_disable(DAC0, DAC_OUT0);
```

dac_output_buffer_enable

The description of dac_output_buffer_enable is shown as below:

Table 3-149. Function dac_output_buffer_enable

Function name	dac_output_buffer_enable
Function prototype	void dac_output_buffer_enable(uint32_t dac_periph, uint8_t dac_out);
Function descriptions	enable DAC output buffer
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC0_OUT0 output buffer */
```

```
dac_output_buffer_enable(DAC0, DAC_OUT0);
```

dac_output_buffer_disable

The description of dac_output_buffer_disable is shown as below:

Table 3-150. Function dac_output_buffer_disable

Function name	dac_output_buffer_disable
Function prototype	void dac_output_buffer_disable(uint32_t dac_periph, uint8_t dac_out);
Function descriptions	disable DAC output buffer
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)

Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC0_OUT0 output buffer */
dac_output_buffer_disable(DAC0, DAC_OUT0);
```

dac_output_value_get

The description of dac_output_value_get is shown as below:

Table 3-151. 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	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)
Output parameter{out}	
-	-
Return value	
uint16_t	DAC output data (0~4095)

Example:

```
/* get the DAC0_OUT0 last data output value */
uint16_t data = 0;
data = dac_output_value_get(DAC0, DAC_OUT0);
```

dac_data_set

The description of dac_data_set is shown as below:

Table 3-152. Function dac_data_set

Function name	dac_data_set
Function prototype	void dac_data_set(uint32_t dac_periph, uint8_t dac_out, uint32_t dac_align, uint16_t data);
Function descriptions	set DAC data holding register value
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)
Input parameter{in}	
dac_align	DAC data alignment mode
<i>DAC_ALIGN_12B_R</i>	12-bit right-aligned data
<i>DAC_ALIGN_12B_L</i>	12-bit left-aligned data
<i>DAC_ALIGN_8B_R</i>	8-bit right-aligned data
Input parameter{in}	
data	data to be loaded (0~4095)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set DAC0_OUT0 data holding register value */
```

```
dac_data_set(DAC0, DAC_OUT0, DAC_ALIGN_8B_R, 0xFF);
```

dac_trigger_enable

The description of dac_trigger_enable is shown as below:

Table 3-153. Function dac_trigger_enable

Function name	dac_trigger_enable
Function prototype	void dac_trigger_enable(uint32_t dac_periph, uint8_t dac_out);
Function descriptions	enable DAC trigger
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Input parameter{in}	
dac_out	DAC output

<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC0_OUT0 trigger */
dac_trigger_enable(DAC0, DAC_OUT0);
```

dac_trigger_disable

The description of `dac_trigger_disable` is shown as below:

Table 3-154. Function `dac_trigger_disable`

Function name	<code>dac_trigger_disable</code>
Function prototype	<code>void dac_trigger_disable(uint32_t dac_periph, uint8_t dac_out);</code>
Function descriptions	disable DAC trigger
Precondition	-
The called functions	-
Input parameter{in}	
<code>dac_periph</code>	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Input parameter{in}	
<code>dac_out</code>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC0_OUT0 trigger */
dac_trigger_disable(DAC0, DAC_OUT0);
```

dac_trigger_source_config

The description of `dac_trigger_source_config` is shown as below:

Table 3-155. Function `dac_trigger_source_config`

Function name	<code>dac_trigger_source_config</code>
Function prototype	<code>void dac_trigger_source_config(uint32_t dac_periph, uint8_t dac_out, uint32_t triggersource);</code>
Function descriptions	configure DAC trigger source

Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
Input parameter{in}	
triggersource	external trigger of DAC
<i>DAC_TRIGGER_T5_TRG</i> 0	TIMER5 TRGO
<i>DAC_TRIGGER_T2_TRG</i> 0	TIMER2 TRGO
<i>DAC_TRIGGER_T6_TRG</i> 0	TIMER6 TRGO
<i>DAC_TRIGGER_T4_TRG</i> 0	TIMER4 TRGO
<i>DAC_TRIGGER_T1_TRG</i> 0	TIMER1 TRGO
<i>DAC_TRIGGER_T3_TRG</i> 0	TIMER3 TRGO
<i>DAC_TRIGGER_EXTI_9</i>	EXTI interrupt line9 event
<i>DAC_TRIGGER_SOFTWARE</i>	software trigger
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DAC0_OUT0 trigger source */
```

```
dac_trigger_source_config(DAC0, DAC_OUT0, DAC_TRIGGER_T1_TRGO);
```

dac_software_trigger_enable

The description of `dac_trigger_source_enable` is shown as below:

Table 3-156. Function `dac_software_trigger_enable`

Function name	<code>dac_software_trigger_enable</code>
Function prototype	<code>void dac_software_trigger_enable(uint32_t dac_periph, uint8_t dac_out);</code>
Function descriptions	enable DAC software trigger

Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC0_OUT0 software trigger */
```

```
dac_software_trigger_enable(DAC0, DAC_OUT0);
```

dac_wave_mode_config

The description of dac_wave_mode_config is shown as below:

Table 3-157. Function dac_wave_mode_config

Function name	dac_wave_mode_config
Function prototype	void dac_wave_mode_config(uint32_t dac_periph, uint8_t dac_out, uint32_t wave_mode);
Function descriptions	configure DAC wave mode
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
Input parameter{in}	
wave_mode	DAC wave mode
<i>DAC_WAVE_DISABLE</i>	wave mode 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_OUT0 wave mode */  
  
dac_wave_mode_config(DAC0, DAC_OUT0, DAC_WAVE_DISABLE);
```

dac_lfsr_noise_config

The description of `dac_lfsr_noise_config` is shown as below:

Table 3-158. Function `dac_lfsr_noise_config`

Function name	<code>dac_lfsr_noise_config</code>
Function prototype	<code>void dac_lfsr_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t unmask_bits);</code>
Function descriptions	configure DAC LFSR noise mode
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
Input parameter{in}	
unmask_bits	LFSR noise unmask bits
<i>DAC_LFSR_BIT0</i>	unmask the LFSR bit0
<i>DAC_LFSR_BITSx_0</i>	unmask the LFSR bits [x:0] (x = 1,2,3..11)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DAC0_OUT0 LFSR noise mode */  
  
dac_lfsr_noise_config(DAC0, DAC_OUT0, DAC_LFSR_BIT0);
```

dac_triangle_noise_config

The description of `dac_triangle_noise_config` is shown as below:

Table 3-159. Function `dac_triangle_noise_config`

Function name	<code>dac_triangle_noise_config</code>
Function prototype	<code>void dac_triangle_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t amplitude);</code>

Function descriptions	configure DAC triangle noise mode
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
Input parameter{in}	
amplitude	the amplitude of the triangle
<i>DAC_TRIANGLE_AMPLIT</i> <i>UDE_x</i>	$x = 2^n - 1$ (n = 1..12)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DAC0_OUT0 triangle noise mode */
```

```
dac_triangle_noise_config(DAC0, DAC_OUT0, DAC_TRIANGLE_AMPLITUDE_1);
```

dac_concurrent_enable

The description of dac_concurrent_enable is shown as below:

Table 3-160. Function dac_concurrent_enable

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

Example:

```
/* enable DAC0 concurrent mode */
```

```
dac_concurrent_enable(DAC0);
```

dac_concurrent_disable

The description of `dac_concurrent_disable` is shown as below:

Table 3-161. Function `dac_concurrent_disable`

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

Example:

```
/* disable DAC0 concurrent mode */
```

```
dac_concurrent_disable(DAC0);
```

dac_concurrent_software_trigger_enable

The description of `dac_concurrent_software_trigger_enable` is shown as below:

Table 3-162. Function `dac_concurrent_software_trigger_enable`

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

Example:

```
/* enable DAC0 concurrent software trigger */
```

```
dac_concurrent_software_trigger_enable(DAC0);
```

dac_concurrent_output_buffer_enable

The description of dac_concurrent_output_buffer_enable is shown as below:

Table 3-163. Function dac_concurrent_output_buffer_enable

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

Example:

```
/* enable DAC0 concurrent buffer function */
dac_concurrent_output_buffer_enable(DAC0);
```

dac_concurrent_output_buffer_disable

The description of dac_concurrent_output_buffer_disable is shown as below:

Table 3-164. Function dac_concurrent_output_buffer_disable

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

Example:

```
/* disable DAC0 concurrent buffer function */
dac_concurrent_output_buffer_disable(DAC0);
```

dac_concurrent_data_set

The description of `dac_concurrent_data_set` is shown as below:

Table 3-165. Function `dac_concurrent_data_set`

Function name	<code>dac_concurrent_data_set</code>
Function prototype	<code>void dac_concurrent_data_set(uint32_t dac_periph, 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}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
dac_align	DAC data alignment mode
<i>DAC_ALIGN_12B_R</i>	12-bit right-aligned data
<i>DAC_ALIGN_12B_L</i>	12-bit left-aligned data
<i>DAC_ALIGN_8B_R</i>	8-bit right-aligned data
Input parameter{in}	
data0	DACx_OUT0 data to be loaded (0~4095)
Input parameter{in}	
data1	DACx_OUT1 data to be loaded (0~4095)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set DAC0 concurrent mode data holding register value */
```

```
dac_concurrent_data_set(DAC0, DAC_ALIGN_8B_R, 0xFF, 0xFF);
```

3.8. DBG

The DBG hold unit helps debugger to debug power saving mode. The DBG registers are listed in chapter [3.8.1](#). the DBG firmware functions are introduced in chapter [3.8.2](#).

3.8.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

Table 3-166. DBG Registers

Registers	Descriptions
DBG_ID	DBG ID code register
DBG_CTL	DBG control register

3.8.2. Descriptions of Peripheral functions

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

Table 3-167. DBG firmware function

Function name	Function description
dbg_id_get	read DBG_ID 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
dbg_trace_pin_mode_set	set trace pin mode

Enum dbg_periph_enum

Table 3-168. 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_TIMER7_HOLD	hold TIMER7 counter 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_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

Member name	Function description
DBG_TIMER10_HOLD	hold TIMER10 counter when core is halted
DBG_I2C2_HOLD	hold I2C2 smbus when core is halted

dbg_id_get

The description of dbg_id_get is shown as below:

Table 3-169. Function dbg_id_get

Function name	dbg_id_get
Function prototype	uint32_t dbg_id_get(void);
Function descriptions	read DBG_ID 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 3-170. Function dbg_low_power_enable

Function name	dbg_low_power_enable
Function prototype	void dbg_low_power_enable(uint32_t dbg_low_power);
Function descriptions	enable 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:

```
/* 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 3-171. Function dbg_low_power_disable

Function name	dbg_low_power_disable
Function prototype	void dbg_low_power_disable(uint32_t dbg_low_power);
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 3-172. 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 3-168. Enum dbg_periph_enum
<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>	hold CANx counter when core is halted (x=0, 1)
<i>DBG_I2Cx_HOLD</i>	hold I2Cx smbus when core is halted (x=0, 1, 2)
<i>DBG_TIMERx_HOLD</i>	hold TIMERx counter when core is halted (x=0~13)
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 3-173. 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}	
dbg_periph	peripheral refer to Table 3-168. Enum dbg_periph_enum
<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>	hold CANx counter when core is halted (x=0, 1)
<i>DBG_I2Cx_HOLD</i>	hold I2Cx smbus when core is halted (x=0, 1, 2)
<i>DBG_TIMERx_HOLD</i>	hold TIMERx counter when core is halted (x=0~13)
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 3-174. 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 3-175. 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();
```

dbg_trace_pin_mode_set

The description of dbg_trace_pin_mode_set is shown as below:

Table 3-176. Function dbg_trace_pin_mode_set

Function name	dbg_trace_pin_mode_set
Function prototype	void dbg_trace_pin_mode_set(uint32_t trace_mode);
Function descriptions	trace pin mode selection
Precondition	-
The called functions	-
Input parameter{in}	
trace_mode	trace pin mode selection
TRACE_MODE_ASYNC	trace pin used for asynchronization mode
TRACE_MODE_SYNC_DATASIZE_1	trace pin used for synchronization mode and data size is 1
TRACE_MODE_SYNC_DATASIZE_2	trace pin used for synchronization mode and data size is 2
TRACE_MODE_SYNC_DATASIZE_4	trace pin used for synchronization mode and data size is 4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* trace pin mode selection */
```

```
dbg_trace_pin_mode_set(TRACE_MODE_ASYNC);
```

3.9. 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 [3.9.1](#). the DCI firmware functions are introduced in chapter [3.9.2](#).

3.9.1. Descriptions of Peripheral registers

DCI registers are listed in the table shown as below:

Table 3-177. DCI Registers

Registers	Descriptions
DCI_CTL	DCI control register
DCI_STAT0	DCI status register 0

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

3.9.2. Descriptions of Peripheral functions

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

Table 3-178. DCI firmware function

Function name	Function description
dc_i_deinit	DCI deinit
dc_i_init	initialize DCI registers
dc_i_enable	enable DCI function
dc_i_disable	disable DCI function
dc_i_capture_enable	enable DCI capture
dc_i_capture_disable	disable DCI capture
dc_i_jpeg_enable	enable DCI jpeg mode
dc_i_jpeg_disable	disable DCI jpeg mode
dc_i_crop_window_enable	enable cropping window function
dc_i_crop_window_disable	disable cropping window function
dc_i_crop_window_config	config DCI cropping window
dc_i_embedded_sync_enable	enable embedded synchronous mode
dc_i_embedded_sync_disable	disable embedded synchronous mode
dc_i_sync_codes_config	config synchronous codes in embedded synchronous mode
dc_i_sync_codes_unmask_config	config synchronous codes unmask in embedded synchronous mode
dc_i_data_read	read DCI data register
dc_i_flag_get	get specified flag
dc_i_interrupt_enable	enable specified DCI interrupt
dc_i_interrupt_disable	disable specified DCI interrupt
dc_i_interrupt_flag_get	get specified interrupt flag
dc_i_interrupt_flag_clear	clear specified interrupt flag

Structure dci_parameter_struct

Table 3-179. 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 3-180. 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 3-181. 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 struct, refer to Table 3-179. Structure dci_parameter_struct
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 3-182. 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 */
```

```
dc_i_enable();
```

dc_i_disable

The description of dc_i_disable is shown as below:

Table 3-183. Function dc_i_disable

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

Example:

```
/* disable DCI function */
```

```
dc_i_disable();
```

dc_i_capture_enable

The description of dc_i_capture_enable is shown as below:

Table 3-184. Function dc_i_capture_enable

Function name	dc_i_capture_enable
Function prototype	void dc_i_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 3-185. 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 3-186. 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 */
```

```
dcj_jpeg_enable();
```

dcj_jpeg_disable

The description of dcj_jpeg_disable is shown as below:

Table 3-187. Function dcj_jpeg_disable

Function name	dcj_jpeg_disable
Function prototype	void dcj_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 */
```

```
dcj_jpeg_disable();
```

dcj_crop_window_enable

The description of dcj_crop_window_enable is shown as below:

Table 3-188. Function dcj_crop_window_enable

Function name	dcj_crop_window_enable
Function prototype	void dcj_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 */
```

```
dc_i_cro_p_window_enable();
```

dc_i_cro_p_window_disable

The description of dc_i_cro_p_window_disable is shown as below:

Table 3-189. Function dc_i_cro_p_window_disable

Function name	dc_i_cro_p_window_disable
Function prototype	void dc_i_cro_p_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 */
```

```
dc_i_cro_p_window_disable();
```

dc_i_cro_p_window_config

The description of dc_i_cro_p_window_config is shown as below:

Table 3-190. Function dc_i_cro_p_window_config

Function name	dc_i_cro_p_window_config
Function prototype	void dc_i_cro_p_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 3-191. 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 3-192. 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 3-193. 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 3-194. 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);
```

dc_i_data_read

The description of dc_i_data_read is shown as below:

Table 3-195. Function dc_i_data_read

Function name	dc_i_data_read
Function prototype	uint32_t dc_i_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 = 0;
data = dc_i_data_read();
```

dc_i_flag_get

The description of dc_i_flag_get is shown as below:

Table 3-196. Function dc_i_flag_get

Function name	dc_i_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	SET or RESET

Example:

```
/* get specified flag */
```

```
FlagStatus status = RESET;
status = dci_flag_get (DCI_FLAG_HS);
```

dc_i_interrupt_enable

The description of dc_i_interrupt_enable is shown as below:

Table 3-197. Function dc_i_interrupt_enable

Function name	dc_i_interrupt_enable
Function prototype	void dc_i_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 3-198. 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
DCI_INT_EF	end of frame interrupt
DCI_INT_OVR	FIFO overrun interrupt
DCI_INT_ESE	embedded synchronous error interrupt
DCI_INT_VSYNC	vsync interrupt
DCI_INT_EL	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 3-199. 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</i> <i>C</i>	vsync interrupt flag
<i>DCI_INT_FLAG_EL</i>	end of line interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get specified interrupt flag */
```

```
FlagStatus status = RESET;
```

```
status = dci_interrupt_flag_get(DCI_INT_FLAG_EF);
```

dc_i_interrupt_flag_clear

The description of dci_interrupt_flag_clear is shown as below:

Table 3-200. 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</i> <i>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 */
```

```
dc_i_interrupt_flag_clear(DCI_INT_FLAG_EF);
```

3.10. 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 [3.10.1](#), the DMA firmware functions are introduced in chapter [3.10.2](#).

3.10.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

Table 3-201. DMA Registers

Registers	Descriptions
DMA_INTF	interrupt flag register
DMA_INTC	interrupt flag clear register
DMA_CHxCTL (x=0..6)	channel x control register
DMA_CHxCNT (x=0..6)	channel x counter register
DMA_CHxPADDR (x=0..6)	channel x peripheral base address register
DMA_CHxMADDR (x=0..6)	channel x memory base address register
DMA_ACFG	additional configuration register

3.10.2. Descriptions of Peripheral functions

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

Table 3-202. DMA firmware function

Function name	Function description
dma_deinit	deinitialize DMA a channel registers
dma_struct_para_init	initialize the parameters of DMA struct with the default values
dma_init	initialize DMA channel
dma_circulation_enable	enable DMA circulation mode
dma_circulation_disable	disable DMA circulation mode
dma_memory_to_memory_enable	enable memory to memory mode
dma_memory_to_memory_disable	disable memory to memory mode
dma_channel_enable	enable DMA channel
dma_channel_disable	disable DMA channel
dma_periph_address_config	set DMA peripheral base address

Function name	Function description
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_width_config	configure transfer data size of memory
dma_periph_width_config	configure transfer data size of peripheral
dma_memory_increase_enable	enable next address increasement algorithm of memory
dma_memory_increase_disable	disable next address increasement algorithm of memory
dma_periph_increase_enable	enable next address increasement algorithm of peripheral
dma_periph_increase_disable	disable next address increasement algorithm of peripheral
dma_transfer_direction_config	configure the direction of data transfer on the channel
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 and interrupt enable bit 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
dma_1_channel_5_fulldata_transfer_enable	enable the DMA1 channel 5 Full_Data transfer mode
dma_1_channel_5_fulldata_transfer_disable	disable the DMA1 channel 5 Full_Data transfer mode

Structure dma_parameter_struct

Table 3-203. Structure dma_parameter_struct

Member name	Function description
periph_addr	peripheral base address
periph_width	transfer data size of peripheral
memory_addr	memory base address
memory_width	transfer data size of memory
number	channel transfer number
priority	channel priority level
periph_inc	peripheral increasing mode
memory_inc	memory increasing mode
direction	channel data transfer direction

dma_deinit

The description of dma_deinit is shown as below:

Table 3-204. Function dma_deinit

Function name	dma_deinit
Function prototype	void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);
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(DMA0: x=0..6; DMA1: x=0..6)	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

dma_struct_para_init

The description of dma_struct_para_init is shown as below:

Table 3-205. Function dma_struct_para_init

Function name	dma_struct_para_init
Function prototype	void dma_struct_para_init(dma_parameter_struct* init_struct);
Function descriptions	Initialize the parameters of DMA struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
Init_struct	Structure for initialization, the structure members can refer to Table 3-203. Structure dma_parameter_struct
Return value	
-	-

Example:

```
/* initialize the parameters of DMA structure with the default values */
dma_parameter_struct init_struct;
```

```
dma_struct_para_init (&init_struct);
```

dma_init

The description of dma_init is shown as below:

Table 3-206. Function dma_init

Function name	<code>dma_init</code>
Function prototype	<code>void dma_init(uint32_t dma_periph, dma_channel_enum channelx, dma_parameter_struct init_struct);</code>
Function descriptions	initialize 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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Input parameter{in}	
init_struct	Structure for initialization, the structure members can refer to Table 3-203. Structure dma_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize DMA0 channel0 */
```

```
dma_parameter_struct dma_init_struct;
```

```
dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
```

```
dma_init_struct.memory_addr = (uint32_t)g_destbuf;
```

```
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
```

```
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_8BIT;
```

```
dma_init_struct.number = TRANSFER_NUM;
```

```
dma_init_struct.periph_addr = (uint32_t)BANK0_WRITE_START_ADDR;
```

```
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
```

```
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_8BIT;
```

```
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
```

```
dma_init(DMA0, DMA_CH0, &dma_init_struct);
```

dma_circulation_enable

The description of dma_circulation_enable is shown as below:

Table 3-207. 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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA0 channel0 mode configuration */
```

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

dma_circulation_disable

The description of dma_circulation_disable is shown as below:

Table 3-208. Function dma_circulation_disable

Function name	dma_circulation_disable
Function prototype	void 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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA0 channel0 mode configuration */
dma_circulation_disable(DMA0, DMA_CH0);
```

dma_memory_to_memory_enable

The description of dma_memory_to_memory_enable is shown as below:

Table 3-209. Function dma_memory_to_memory_enable

Function name	dma_memory_to_memory_enable
Function prototype	void dma_memory_to_memory_enable(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	enable memory to memory 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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA0 channel0 mode configuration */
dma_memory_to_memory_enable(DMA0, DMA_CH0);
```

dma_memory_to_memory_disable

The description of dma_memory_to_memory_disable is shown as below:

Table 3-210. Function dma_memory_to_memory_disable

Function name	dma_memory_to_memory_disable
Function prototype	void dma_memory_to_memory_disable(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	disable memory to memory 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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA0 channel0 mode configuration */
```

```
dma_memory_to_memory_enable(DMA0, DMA_CH0);
```

dma_channel_enable

The description of dma_channel_enable is shown as below:

Table 3-211. 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
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection

6; DMA1: x=0..6)	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA0 channel0 transfer */
```

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

dma_channel_disable

The description of dma_channel_disable is shown as below:

Table 3-212. Function dma_channel_disable

Function name	dma_channel_disable
Function prototype	void dma_channel_disable(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	disable DMA channel
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(DMA0:x=0..6; DMA1: x=0..6)	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA0 channel0 transfer */
```

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

dma_periph_address_config

The description of dma_periph_address_config is shown as below:

Table 3-213. Function dma_periph_address_config

Function name	dma_periph_address_config
Function prototype	void dma_periph_address_config(uint32_t dma_periph,

	<code>dma_channel_enum channelx, uint32_t address);</code>
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(DMA0: x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Input parameter{in}	
address	peripheral base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA0 channel0 address */

#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 3-214. Function `dma_memory_address_config`

Function name	<code>dma_memory_address_config</code>
Function prototype	<code>void dma_memory_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);</code>
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(DMA0: x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Input parameter{in}	
address	memory base address

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA0 channel0 memory address */
uint8_t g_destbuf[TRANSFER_NUM];

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

dma_transfer_number_config

The description of dma_transfer_number_config is shown as below:

Table 3-215. 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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Input parameter{in}	
number	data transfer number
<i>0x0000-0xffff</i>	number
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA0 channel0 transfer number */

#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 3-216. 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
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
uint32_t	0x0000-0xffff

Example:

```
/* get DMA0 channel0 transfer number */
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 3-217. 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(DMA0:x=0..6; DMA1: x=0..6)</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:

```
/* configure DMA0 channel0 priority */
```

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

dma_memory_width_config

The description of dma_memory_width_config is shown as below:

Table 3-218. 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 mwidth);
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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Input parameter{in}	
mwidth	transfer data width of memory
<i>DMA_MEMORY_WIDTH_8BIT</i>	transfer data width of memory is 8-bit
<i>DMA_MEMORY_WIDTH_16BIT</i>	transfer data width of memory is 16-bit

<i>H_16BIT</i>	
<i>DMA_MEMORY_WIDT</i> <i>H_32BIT</i>	transfer data width of memory is 32-bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA0 channel0 memory width */
```

```
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 3-219. Function dma_periph_width_config

Function name	dma_periph_width_config
Function prototype	void dma_periph_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t pwidth);
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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Input parameter{in}	
pwidth	transfer data width of peripheral
<i>DMA_PERIPHERAL_WIDTH_8BIT</i>	transfer data width of peripheral is 8-bit
<i>DMA_PERIPHERAL_WIDTH_16BIT</i>	transfer data width of peripheral is 16-bit
<i>DMA_PERIPHERAL_WIDTH_32BIT</i>	transfer data width of peripheral is 32-bit
Output parameter{out}	
-	-
Return value	
-	-

Example:


```
/* configure DMA0 channel0 peripheral width */
```

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

dma_memory_increase_enable

The description of dma_memory_increase_enable is shown as below:

Table 3-220. Function dma_memory_increase_enable

Function name	dma_memory_increase_enable
Function prototype	void dma_memory_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	enable next address increasement algorithm 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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA0 channel0 memory increase */
```

```
dma_memory_increase_enable(DMA0, DMA_CH0);
```

dma_memory_increase_disable

The description of dma_memory_increase_disable is shown as below:

Table 3-221. Function dma_memory_increase_disable

Function name	dma_memory_increase_disable
Function prototype	void dma_memory_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	disable next address increasement algorithm 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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA0 channel0 memory increase */
dma_memory_increase_disable(DMA0, DMA_CH0);
```

dma_periph_increase_enable

The description of dma_periph_increase_enable is shown as below:

Table 3-222. Function dma_periph_increase_enable

Function name	dma_periph_increase_enable
Function prototype	void dma_periph_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	enable next address increasement algorithm 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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA0 channel0 peripheral increase */
dma_periph_increase_enable(DMA0, DMA_CH0);
```

dma_periph_increase_disable

The description of dma_periph_increase_disable is shown as below:

Table 3-223. Function dma_periph_increase_disable

Function name	dma_periph_increase_disable
Function prototype	void dma_periph_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	disable next address increasement algorithm 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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA0 channel0 peripheral increase */
```

```
dma_periph_increase_disable(DMA0, DMA_CH0);
```

dma_transfer_direction_config

The description of dma_transfer_direction_config is shown as below:

Table 3-224. Function dma_transfer_direction_config

Function name	dma_transfer_direction_config
Function prototype	void dma_transfer_direction_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t direction);
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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Input parameter{in}	
direction	specify the direction of data transfer

<i>DMA_PERIPHERAL_TO_MEMORY</i>	read from peripheral and write to memory
<i>DMA_MEMORY_TO_PERIPHERAL</i>	read from memory and write to peripheral
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config DMA0 channel0 peripheral to memory */
dma_transfer_direction_config(DMA0, DMA_CH0, DMA_PERIPHERAL_TO_MEMORY);
```

dma_1_channel_5_fulldata_transfer_enable

The description of dma_1_channel_5_fulldata_transfer_enable is shown as below:

Table 3-225. Function dma_1_channel_5_fulldata_transfer_enable

Function name	dma_1_channel_5_fulldata_transfer_enable
Function prototype	void dma_1_channel_5_fulldata_transfer_enable(void);
Function descriptions	enable the DMA1 channel 5 Full_Data transfer mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the DMA1 channel 5 Full_Data transfer mode */
dma_1_channel_5_fulldata_transfer_enable();
```

dma_1_channel_5_fulldata_transfer_disable

The description of dma_1_channel_5_fulldata_transfer_disable is shown as below:

Table 3-226. Function dma_1_channel_5_fulldata_transfer_disable

Function name	dma_1_channel_5_fulldata_transfer_disable
Function prototype	void dma_1_channel_5_fulldata_transfer_disable(void);
Function descriptions	disable the DMA1 channel 5 Full_Data transfer mode
Precondition	-
The called functions	-

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

Example:

```
/* disable the DMA1 channel 5 Full_Data transfer mode */
```

```
dma_1_channel_5_fulldata_transfer_disable();
```

dma_flag_get

The description of dma_flag_get is shown as below:

Table 3-227. Function dma_flag_get

Function name	dma_flag_get
Function prototype	FlagStatus dma_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
Function descriptions	check 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(DMA0: x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Input parameter{in}	
flag	specify get which flag
<i>DMA_FLAG_G</i>	global interrupt flag of channel
<i>DMA_FLAG_FTF</i>	full transfer finish flag of channel
<i>DMA_FLAG_HTF</i>	half transfer finish flag of channel
<i>DMA_FLAG_ERR</i>	error flag of channel
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get DMA0 channel0 flag */
```

```
FlagStatus flag = RESET;
```

```
flag = dma_flag_get(DMA0, DMA_CH0, DMA_FLAG_FTF);
```

dma_flag_clear

The description of dma_flag_clear is shown as below:

Table 3-228. Function dma_flag_clear

Function name	dma_flag_clear
Function prototype	void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
Function descriptions	clear DMA a channel 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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Input parameter{in}	
flag	specify get which flag
<i>DMA_FLAG_G</i>	global interrupt flag of channel
<i>DMA_FLAG_FTF</i>	full transfer finish flag of channel
<i>DMA_FLAG_HTF</i>	half transfer finish flag of channel
<i>DMA_FLAG_ERR</i>	error flag of channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear DMA0 channel0 flag */
```

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

dma_interrupt_enable

The description of dma_interrupt_enable is shown as below:

Table 3-229. 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
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Input parameter{in}	
source	DMA interrupt source
<i>DMA_INT_FTF</i>	full transfer finish interrupt of channel
<i>DMA_INT_HTF</i>	half transfer finish interrupt of channel
<i>DMA_INT_ERR</i>	error interrupt of channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA0 channel0 interrupt configuration */
```

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

dma_interrupt_disable

The description of dma_interrupt_disable is shown as below:

Table 3-230. 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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Input parameter{in}	
source	DMA interrupt source

<i>DMA_INT_FTF</i>	full transfer finish interrupt of channel
<i>DMA_INT_HTF</i>	half transfer finish interrupt of channel
<i>DMA_INT_ERR</i>	error interrupt of channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA0 channel0 interrupt configuration */
```

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

dma_interrupt_flag_get

The description of dma_interrupt_flag_get is shown as below:

Table 3-231. 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 flag);
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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Input parameter{in}	
flag	specify get which flag
<i>DMA_INT_FLAG_G</i>	global interrupt flag of channel
<i>DMA_INT_FLAG_FTF</i>	full transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_HTF</i>	half transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_ERR</i>	error interrupt flag of channel
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get DMA0 channel3 interrupt flag */
```



```
FlagStatus int_flag_status = RESET;
```

```
int_flag_status = dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_G);
```

dma_interrupt_flag_clear

The description of dma_interrupt_flag_clear is shown as below:

Table 3-232. 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 flag);
Function descriptions	clear DMA a channel 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(DMA0:x=0..6; DMA1: x=0..6)</i>	DMA channel selection
Input parameter{in}	
flag	specify get which flag
<i>DMA_INT_FLAG_G</i>	global interrupt flag of channel
<i>DMA_INT_FLAG_FTF</i>	full transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_HTF</i>	half transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_ERR</i>	error interrupt flag of channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear DMA0 channel3 interrupt flag */
```

```
dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
```

3.11. 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 [3.11.1](#), the ENET firmware functions are introduced in chapter [3.11.2](#).

3.11.1. Descriptions of Peripheral registers

ENET registers are listed in the table shown as below:

Table 3-233. 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_FCTH	MAC flow control threshold 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_INTF	MAC interrupt flag register
ENET_MAC_INTMSK	MAC interrupt mask register
ENET_MAC_ADDR0H	MAC address 0 high register
ENET_MAC_ADDR0L	MAC address 0 low register
ENET_MAC_ADDR1H	MAC address 1 high register
ENET_MAC_ADDR1L	MAC address 1 low register
ENET_MAC_ADDT2H	MAC address 2 high register
ENET_MAC_ADDR2L	MAC address 2 low register
ENET_MAC_ADDR3H	MAC address 3 high register
ENET_MAC_ADDR3L	MAC address 3 low register
ENET_MSC_CTL	MSC control register
ENET_MSC_RINTF	MSC receive interrupt flag register

Registers	Descriptions
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
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_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_TDTA 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_CTDA DDR	DMA current transmit descriptor address register
ENET_DMA_CRDA	DMA current receive descriptor address register

Registers	Descriptions
DDR	
ENET_DMA_CTBA DDR	DMA current transmit buffer address register
ENET_DMA_CRBA DDR	DMA current receive buffer address register

3.11.2. Descriptions of Peripheral functions

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

Table 3-234. 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
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)

Function name	Function description
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_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_fliter_feature_enable	enable ENET fliter feature
enet_fliter_feature_disable	disable ENET fliter 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
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_desc_receive_complete_bit_enable	when receiving the completed, set RS bit in ENET_DMA_STAT register will immediately set

Function name	Function description
enet_desc_receive_complete_bit_disable	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
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_get	get MAC statistics counter
PTP function	
enet_ptp_subsecond_2_nanosecond	change subsecond to nanosecond
enet_ptp_nanosecond_2_subsecond	change nanosecond to subsecond
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_start	configure and start PTP timestamp counter
enet_ptp_finecorrection_adjfreq	adjust frequency in fine method by configure addend register
enet_ptp_coarsecorrection_systime_update	update system time in coarse method

Function name	Function description
enet_ptp_finecorrection_settime	set system time in fine method
enet_ptp_flag_get	get the ptp flag status
Other	
enet_initpara_reset	reset the ENET initpara struct, call it before using enet_initpara_config()

Structure enet_descriptors_struct

Table 3-235. 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

Structure enet_ptp_systime_struct

Table 3-236. Structure enet_ptp_systime_struct

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

Enum enet_flag_enum

Table 3-237. Enum enet_flag_enum

member name	Function description
ENET_MAC_FLAG_MPKR	magic packet received flag
ENET_MAC_FLAG_WUFR	wakeup frame received flag
ENET_MAC_FLAG_FLOWCONTROL	flow control status flag
ENET_MAC_FLAG_WUM	WUM status flag
ENET_MAC_FLAG_MSC	MSC status flag
ENET_MAC_FLAG_MSCR	MSC receive status flag
ENET_MAC_FLAG_MSCT	MSC transmit status flag
ENET_MAC_FLAG_	timestamp trigger status flag

member name	Function description
TMST	
ENET_PTP_FLAG_TSSCO	timestamp second counter overflow flag
ENET_PTP_FLAG_TTM	target time match flag
ENET_MSC_FLAG_RFCE	received frames CRC error flag
ENET_MSC_FLAG_RFAE	received frames alignment error flag
ENET_MSC_FLAG_RGUF	received good unicast frames flag
ENET_MSC_FLAG_TGFSC	transmitted good frames single collision flag
ENET_MSC_FLAG_TGFMSC	transmitted good frames more single collision flag
ENET_MSC_FLAG_TGF	transmitted good frames flag
ENET_DMA_FLAG_TS	transmit status flag
ENET_DMA_FLAG_TPS	transmit process stopped status flag
ENET_DMA_FLAG_TBU	transmit buffer unavailable status flag
ENET_DMA_FLAG_TJT	transmit jabber timeout status flag
ENET_DMA_FLAG_RO	receive overflow status flag
ENET_DMA_FLAG_TU	transmit underflow status flag
ENET_DMA_FLAG_RS	receive status flag
ENET_DMA_FLAG_RBU	receive buffer unavailable status flag
ENET_DMA_FLAG_RPS	receive process stopped status flag
ENET_DMA_FLAG_RWT	receive watchdog timeout status flag
ENET_DMA_FLAG_ET	early transmit status flag
ENET_DMA_FLAG_FBE	fatal bus error status flag
ENET_DMA_FLAG_	early receive status flag

member name	Function description
ER	
ENET_DMA_FLAG_AI	abnormal interrupt summary flag
ENET_DMA_FLAG_NI	normal interrupt summary flag
ENET_DMA_FLAG_EB_DMA_ERROR	error during data transfer by RxDMA/TxDMA flag
ENET_DMA_FLAG_EB_TRANSFER_ERROR	error during write/read transfer flag
ENET_DMA_FLAG_EB_ACCESS_ERROR	error during data buffer/descriptor access flag
ENET_DMA_FLAG_MSC	MSC status flag
ENET_DMA_FLAG_WUM	WUM status flag
ENET_DMA_FLAG_TST	timestamp trigger status flag

Enum enet_flag_clear_enum

Table 3-238. Enum enet_flag_clear_enum

member name	Function description
ENET_DMA_FLAG_TS_CLR	transmit status flag clear
ENET_DMA_FLAG_TPS_CLR	transmit process stopped status flag clear
ENET_DMA_FLAG_TBU_CLR	transmit buffer unavailable status flag clear
ENET_DMA_FLAG_TJT_CLR	transmit jabber timeout status flag clear
ENET_DMA_FLAG_RO_CLR	receive overflow status flag clear
ENET_DMA_FLAG_TU_CLR	transmit underflow status flag clear
ENET_DMA_FLAG_RS_CLR	receive status flag clear
ENET_DMA_FLAG_RBU_CLR	receive buffer unavailable status flag clear
ENET_DMA_FLAG_RPS_CLR	receive process stopped status flag clear

member name	Function description
ENET_DMA_FLAG_RWT_CLR	receive watchdog timeout status flag clear
ENET_DMA_FLAG_ET_CLR	early transmit status flag clear
ENET_DMA_FLAG_FBE_CLR	fatal bus error status flag clear
ENET_DMA_FLAG_ER_CLR	early receive status flag clear
ENET_DMA_FLAG_AI_CLR	abnormal interrupt summary flag clear
ENET_DMA_FLAG_NI_CLR	normal interrupt summary flag clear

Enum enet_int_enum

Table 3-239. Enum enet_int_enum

member name	Function description
ENET_MAC_INT_WUMIM	WUM interrupt mask
ENET_MAC_INT_TSTMIM	timestamp trigger interrupt mask
ENET_MSC_INT_RFCEIM	received frame CRC error interrupt mask
ENET_MSC_INT_RFAEIM	received frames alignment error interrupt mask
ENET_MSC_INT_RGUFIM	received good unicast frames interrupt mask
ENET_MSC_INT_TGFSCIM	transmitted good frames single collision interrupt mask
ENET_MSC_INT_TGFMSCIM	transmitted good frames more single collision interrupt mask
ENET_MSC_INT_TGFIM	transmitted good frames interrupt mask
ENET_DMA_INT_TIE	transmit interrupt enable
ENET_DMA_INT_TPSIE	transmit process stopped interrupt enable
ENET_DMA_INT_TBUIE	transmit buffer unavailable interrupt enable
ENET_DMA_INT_TJTIE	transmit jabber timeout interrupt enable
ENET_DMA_INT_R	receive overflow interrupt enable

member name	Function description
<i>OIE</i>	
<i>ENET_DMA_INT_TUIE</i>	transmit underflow interrupt enable
<i>ENET_DMA_INT_RIE</i>	receive interrupt enable
<i>ENET_DMA_INT_RBUIE</i>	receive buffer unavailable interrupt enable
<i>ENET_DMA_INT_RPSIE</i>	receive process stopped interrupt enable
<i>ENET_DMA_INT_RWTIE</i>	receive watchdog timeout interrupt enable
<i>ENET_DMA_INT_ETIE</i>	early transmit interrupt enable
<i>ENET_DMA_INT_FBEIE</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

Enum enet_int_flag_enum

Table 3-240. Enum enet_int_flag_enum

member name	Function description
<i>ENET_MAC_INT_FLAG_WUM</i>	WUM status flag
<i>ENET_MAC_INT_FLAG_MSC</i>	MSC status flag
<i>ENET_MAC_INT_FLAG_MSCR</i>	MSC receive status flag
<i>ENET_MAC_INT_FLAG_MSCT</i>	MSC transmit status flag
<i>ENET_MAC_INT_FLAG_TMST</i>	time stamp trigger status flag
<i>ENET_MSC_INT_FLAG_RFCE</i>	received frames CRC error flag
<i>ENET_MSC_INT_FLAG_RFAE</i>	received frames alignment error flag
<i>ENET_MSC_INT_FLAG_RGUF</i>	received good unicast frames flag

member name	Function description
<i>ENET_MSC_INT_F</i> <i>LAG_TGFSC</i>	transmitted good frames single collision flag
<i>ENET_MSC_INT_F</i> <i>LAG_TGFMSC</i>	transmitted good frames more single collision flag
<i>ENET_MSC_INT_F</i> <i>LAG_TGF</i>	transmitted good frames flag
<i>ENET_DMA_INT_F</i> <i>LAG_TS</i>	transmit status flag
<i>ENET_DMA_INT_F</i> <i>LAG_TPS</i>	transmit process stopped status flag
<i>ENET_DMA_INT_F</i> <i>LAG_TBU</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_F</i> <i>LAG_TJT</i>	transmit jabber timeout status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RO</i>	receive overflow status flag
<i>ENET_DMA_INT_F</i> <i>LAG_TU</i>	transmit underflow status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RS</i>	receive status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RBU</i>	receive buffer unavailable status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RPS</i>	receive process stopped status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RWT</i>	receive watchdog timeout status flag
<i>ENET_DMA_INT_F</i> <i>LAG_ET</i>	early transmit status flag
<i>ENET_DMA_INT_F</i> <i>LAG_FBE</i>	fatal bus error status flag
<i>ENET_DMA_INT_F</i> <i>LAG_ER</i>	early receive status flag
<i>ENET_DMA_INT_F</i> <i>LAG_AI</i>	abnormal interrupt summary flag
<i>ENET_DMA_INT_F</i> <i>LAG_NI</i>	normal interrupt summary flag
<i>ENET_DMA_INT_F</i> <i>LAG_MSC</i>	MSC status flag
<i>ENET_DMA_INT_F</i> <i>LAG_WUM</i>	WUM status flag
<i>ENET_DMA_INT_F</i> <i>LAG_TST</i>	timestamp trigger status flag

Enum enet_int_flag_clear_enum

Table 3-241. Enum enet_int_flag_clear_enum

member name	Function description
<i>ENET_DMA_INT_F LAG_TS_CLR</i>	transmit status flag
<i>ENET_DMA_INT_F LAG_TPS_CLR</i>	transmit process stopped status flag
<i>ENET_DMA_INT_F LAG_TBU_CLR</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_F LAG_TJT_CLR</i>	transmit jabber timeout status flag
<i>ENET_DMA_INT_F LAG_RO_CLR</i>	receive overflow status flag
<i>ENET_DMA_INT_F LAG_TU_CLR</i>	transmit underflow status flag
<i>ENET_DMA_INT_F LAG_RS_CLR</i>	receive status flag
<i>ENET_DMA_INT_F LAG_RBU_CLR</i>	receive buffer unavailable status flag
<i>ENET_DMA_INT_F LAG_RPS_CLR</i>	receive process stopped status flag
<i>ENET_DMA_INT_F LAG_RWT_CLR</i>	receive watchdog timeout status flag
<i>ENET_DMA_INT_F LAG_ET_CLR</i>	early transmit status flag
<i>ENET_DMA_INT_F LAG_FBE_CLR</i>	fatal bus error status flag
<i>ENET_DMA_INT_F LAG_ER_CLR</i>	early receive status flag
<i>ENET_DMA_INT_F LAG_AI_CLR</i>	abnormal interrupt summary flag
<i>ENET_DMA_INT_F LAG_NI_CLR</i>	normal interrupt summary flag

Enum enet_desc_reg_enum

Table 3-242. Enum enet_desc_reg_enum

member name	Function description
<i>ENET_RX_DESC_T ABLE</i>	the start address of the receive descriptor table
<i>ENET_RX_CURRE NT_DESC</i>	the start descriptor address of the current receive descriptor read by the RxDMA controller
<i>ENET_RX_CURRE</i>	the current receive buffer address being read by the RxDMA controller

member name	Function description
<i>NT_BUFFER</i>	
<i>ENET_TX_DESC_TABLE</i>	the start address of the transmit descriptor table
<i>ENET_TX_CURRENT_DESC</i>	the start descriptor address of the current transmit descriptor read by the TxDMA controller
<i>ENET_TX_CURRENT_BUFFER</i>	the current transmit buffer address being read by the TxDMA controller

Enum enet_msc_counter_enum

Table 3-243. Enum enet_msc_counter_enum

member name	Function description
<i>ENET_MSC_TX_SINGLE_COLL_CNT</i>	MSC transmitted good frames after a single collision counter
<i>ENET_MSC_TX_MULT_COLL_CNT</i>	MSC transmitted good frames after more than a single collision counter
<i>ENET_MSC_TX_GOOD_FRAMES_CNT</i>	MSC transmitted good frames counter
<i>ENET_MSC_RX_CRC_ERR_CNT</i>	MSC received frames with CRC error counter
<i>ENET_MSC_RX_ALIGN_ERR_CNT</i>	MSC received frames with alignment error counter
<i>ENET_MSC_RX_GOOD_UNICAST_FRAMES_CNT</i>	MSC received good unicast frames counter

Enum enet_option_enum

Table 3-244. Enum enet_option_enum

member name	Function description
<i>FORWARD_OPTION</i>	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

member name	Function description
<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

Enum enet_mediamode_enum

Table 3-245. Enum enet_mediamode_enum

member name	Function description
<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_LOOPBACK_MODE</i>	MAC in loopback mode at the MII

Enum enet_chksumconf_enum

Table 3-246. Enum enet_chksumconf_enum

member name	Function description
<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

Enum enet_frmrecept_enum

Table 3-247. Enum enet_frmrecept_enum

member name	Function description
<i>ENET_PROMISCUOUS_MODE</i>	promiscuous mode enabled
<i>ENET_RECEIVEALL</i>	all received frame are forwarded to application

<i>L</i>	
<i>ENET_BROADCAST_T_FRAMES_PASS</i>	the address filters pass all received broadcast frames
<i>ENET_BROADCAST_T_FRAMES_DROP</i>	the address filters filter all incoming broadcast frames

Enum `enet_registers_type_enum`

Table 3-248. Enum `enet_registers_type_enum`

member name	Function description
<i>ALL_MAC_REG</i>	get the registers within the offset scope range <code>ENET_MAC_CFG</code> to <code>ENET_MAC_FCTH</code>
<i>ALL_MSC_REG</i>	get the registers within the offset scope range <code>ENET_MSC_CTL</code> to <code>ENET_MSC_RGUFCNT</code>
<i>ALL_PTP_REG</i>	get the registers within the offset scope range <code>ENET_PTP_TSCTL</code> to <code>ENET_PTP_PPSCTL</code>
<i>ALL_DMA_REG</i>	get the registers within the offset scope range <code>ENET_DMA_BCTL</code> to <code>ENET_DMA_CRBADDR</code>

Enum `enet_dmadirection_enum`

Table 3-249. Enum `enet_dmadirection_enum`

member name	Function description
<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors

Enum `enet_phydirection_enum`

Table 3-250. Enum `enet_phydirection_enum`

member name	Function description
<i>ENET_PHY_WRITE</i>	write data to phy register
<i>ENET_PHY_READ</i>	read data from phy register

Enum `enet_regdirection_enum`

Table 3-251. Enum `enet_regdirection_enum`

member name	Function description
<i>ENET_REG_READ</i>	read register
<i>ENET_REG_WRITE</i>	write register

Enum `enet_macaddress_enum`

Table 3-252. Enum `enet_macaddress_enum`

member name	Function description
<i>ENET_MAC_ADDR</i>	set MAC address 0 filter

<i>ESS0</i>	
<i>ENET_MAC_ADDR</i> <i>ESS1</i>	set MAC address 1 filter
<i>ENET_MAC_ADDR</i> <i>ESS2</i>	set MAC address 2 filter
<i>ENET_MAC_ADDR</i> <i>ESS3</i>	set MAC address 3 filter

Enum enet_descstate_enum

Table 3-253. Enum enet_descstate_enum

member name	Function description
<i>TXDESC_COLLISION_COUNT</i>	the number of collisions occurred before the frame was transmitted
<i>TXDESC_BUFFER_1_ADDR</i>	the buffer1 address of the Tx frame
<i>RXDESC_FRAME_LENGTH</i>	the byte length of the received frame that was transferred to the buffer
<i>RXDESC_BUFFER_1_SIZE</i>	receive buffer 1 size
<i>RXDESC_BUFFER_2_SIZE</i>	receive buffer 2 size
<i>RXDESC_BUFFER_1_ADDR</i>	the buffer1 address of the Rx frame

enet_deinit

The description of enet_deinit is shown as below:

Table 3-254. 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 3-255. 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, refer to Table 3-244. Enum enet_option_enum only one parameter can be selected which is shown as below
FORWARD_OPTION	choose to configure the frame forward related parameters
DMABUS_OPTION	choose to configure the DMA bus mode related parameters
DMA_MAXBURST_OPTION	choose to configure the DMA max burst related parameters
DMA_ARBITRATION_OPTION	choose to configure the DMA arbitration related parameters
STORE_OPTION	choose to configure the store forward mode related parameters
DMA_OPTION	choose to configure the DMA related parameters
VLAN_OPTION	choose to configure vlan related parameters
FLOWCTL_OPTION	choose to configure flow control related parameters
HASHH_OPTION	choose to configure hash high
HASHL_OPTION	choose to configure hash low
FILTER_OPTION	choose to configure frame filter related parameters
HALFDUPLEX_OPTION	choose to configure halfduplex mode related parameters
TIMER_OPTION	choose to configure time counter related parameters
INTERFRAMEGAP_OPTION	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 FORWARD_OPTION	
value1	ENET_AUTO_PADCRC_DROP_ENABLE / ENET_AUTO_PADCRC_DROP_DISABLE
value2	ENET_FORWARD_ERRFRAMES_ENABLE /

GD32F20x Firmware Library User Guide

	<i>ENET_FORWARD_ERRFRAMES_DISABLE</i>
value3	<i>ENET_FORWARD_UNDEERSZ_GOODFRAMES_ENABLE /</i> <i>ENET_FORWARD_UNDEERSZ_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>
When value of parameter option is <i>DMA_MAXBURST_OPTION</i>	
value1	<i>ENET_RXDP_1BEAT / ENET_RXDP_2BEAT / ENET_RXDP_4BEAT /</i> <i>ENET_RXDP_8BEAT / ENET_RXDP_16BEAT / ENET_RXDP_32BEAT /</i> <i>ENET_RXDP_4xPGBL_4BEAT / ENET_RXDP_4xPGBL_8BEAT /</i> <i>ENET_RXDP_4xPGBL_16BEAT / ENET_RXDP_4xPGBL_32BEAT /</i> <i>ENET_RXDP_4xPGBL_64BEAT / ENET_RXDP_4xPGBL_128BEAT</i>
value2	<i>ENET_PGBL_1BEAT / ENET_PGBL_2BEAT / ENET_PGBL_4BEAT /</i> <i>ENET_PGBL_8BEAT / ENET_PGBL_16BEAT / ENET_PGBL_32BEAT /</i> <i>ENET_PGBL_4xPGBL_4BEAT / ENET_PGBL_4xPGBL_8BEAT /</i> <i>ENET_PGBL_4xPGBL_16BEAT / ENET_PGBL_4xPGBL_32BEAT /</i> <i>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 / ENET_ARBITRATION_RXTX_1_1 /</i> <i>ENET_ARBITRATION_RXTX_2_1 / ENET_ARBITRATION_RXTX_3_1 /</i> <i>ENET_ARBITRATION_RXTX_4_1</i>
When value of parameter option is <i>STORE_OPTION</i>	
value1	<i>ENET_RX_MODE_STOREFORWARD /</i> <i>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 /</i> <i>ENET_RX_THRESHOLD_96BYTES / ENET_RX_THRESHOLD_128BYTES</i>
value4	<i>ENET_TX_THRESHOLD_64BYTES / ENET_TX_THRESHOLD_128BYTES</i> <i>/ ENET_TX_THRESHOLD_192BYTES /</i> <i>ENET_TX_THRESHOLD_256BYTES / ENET_TX_THRESHOLD_40BYTES</i> <i>/ ENET_TX_THRESHOLD_32BYTES / ENET_TX_THRESHOLD_24BYTES</i> <i>/ ENET_TX_THRESHOLD_16BYTES</i>
When value of parameter option is <i>DMA_OPTION</i>	
value1	<i>ENET_FLUSH_RXFRAME_ENABLE /</i> <i>ENET_FLUSH_RXFRAME_DISABLE</i>
value2	<i>ENET_SECONDFRAME_OPT_ENABLE /</i> <i>ENET_SECONDFRAME_OPT_DISABLE</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>	

GD32F20x Firmware Library User Guide

value1	MAC_FCTL_PTM(regval)
value2	ENET_ZERO_QUANTA_PAUSE_ENABLE / ENET_ZERO_QUANTA_PAUSE_DISABLE
value3	ENET_PAUSETIME_MINUS4 / ENET_PAUSETIME_MINUS28 / ENET_PAUSETIME_MINUS144/ENET_PAUSETIME_MINUS256
value4	ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDetect / ENET_UNIQUE_PAUSEDetect
value5	ENET_RX_FLOWCONTROL_ENABLE / ENET_RX_FLOWCONTROL_DISABLE
value6	ENET_TX_FLOWCONTROL_ENABLE / ENET_TX_FLOWCONTROL_DISABLE
When value of parameter option is HASHH_OPTION	
value1	0x0~0xFFFF FFFFU
When value of parameter option is HASHL_OPTION	
value1	0x0~0xFFFF FFFFU
When value of parameter option is FILTER_OPTION	
value1	ENET_SRC_FILTER_NORMAL_ENABLE / ENET_SRC_FILTER_INVERSE_ENABLE / ENET_SRC_FILTER_DISABLE
value2	ENET_DEST_FILTER_INVERSE_ENABLE / ENET_DEST_FILTER_INVERSE_DISABLE
value3	ENET_MULTICAST_FILTER_HASH_OR_PERFECT / ENET_MULTICAST_FILTER_HASH / ENET_MULTICAST_FILTER_PERFECT / ENET_MULTICAST_FILTER_NONE
value4	ENET_UNICAST_FILTER_EITHER / ENET_UNICAST_FILTER_HASH / ENET_UNICAST_FILTER_PERFECT
value5	ENET_PCFRM_PREVENT_ALL / ENET_PCFRM_PREVENT_PAUSEFRAME / ENET_PCFRM_FORWARD_ALL / ENET_PCFRM_FORWARD_FILTERED
When value of parameter option is HALFDUPLEX_OPTION	
value1	ENET_CARRIERSENSE_ENABLE / ENET_CARRIERSENSE_DISABLE
value2	ENET_RECEIVEOWN_ENABLE / ENET_RECEIVEOWN_DISABLE
value3	ENET_RETRYTRANSMISSION_ENABLE / ENET_RETRYTRANSMISSION_DISABLE
value4	ENET_BACKOFFLIMIT_10 / ENET_BACKOFFLIMIT_8 / ENET_BACKOFFLIMIT_4 / ENET_BACKOFFLIMIT_1
value5	ENET_DEFERRALCHECK_ENABLE / ENET_DEFERRALCHECK_DISABLE
When value of parameter option is TIMER_OPTION	
value1	ENET_WATCHDOG_ENABLE / ENET_WATCHDOG_DISABLE
value2	ENET_JABBER_ENABLE / ENET_JABBER_DISABLE
When value of parameter option is INTERFRAMEGAP_OPTION	
value1	ENET_INTERFRAMEGAP_96BIT / ENET_INTERFRAMEGAP_88BIT /

	<i>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 3-256. Function enet_init

Function name	enet_init
Function prototype	ErrStatus enet_init(enet_mediamode_enum mediamode, enet_chksumconf_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}	
mediamode	PHY mode and mac loopback configurations, refer to Table 3-245. Enum enet_mediamode_enum only one parameter can be selected
ENET_AUTO_NEGOTIATION	PHY auto negotiation
ENET_100M_FULLDUPLEX	100Mbit/s, full-duplex
ENET_100M_HALFDUPLEX	100Mbit/s, half-duplex
ENET_10M_FULLDUPLEX	10Mbit/s, full-duplex
ENET_10M_HALFDUPLEX	10Mbit/s, half-duplex
ENET_LOOPBACKMODE	MAC in loopback mode at the MII
Input parameter{in}	

checksum	IP frame checksum offload function, refer to Table 3-246. Enum <i>enet_chksumconf_enum</i> only one parameter can be selected
<i>ENET_NO_AUTOCHESUM</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}	
recept	frame filter function, refer to Table 3-247. Enum <i>enet_frmrecept_enum</i> only one parameter can be selected
<i>ENET_PROMISCUOUS_MODE</i>	promiscuous mode enabled
<i>ENET_RECEIVEALL</i>	all received frame are forwarded to application
<i>ENET_BROADCAST_FRAMES_PASS</i>	the address filters pass all received broadcast frames
<i>ENET_BROADCAST_FRAMES_DROP</i>	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_DROP_FAILFRAMES, ENET_BROADCAST_FRAMES_PASS);
```

enet_software_reset

The description of enet_software_reset is shown as below:

Table 3-257. 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 3-258. 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 3-259. 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' parameters in chain mode
Precondition	-
The called functions	-

Input parameter{in}	
direction	the descriptors which users want to init, refer to Table 3-249. Enum <i>enet_dmadirection_enum</i> 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' 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 3-260. 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' parameters in ring mode
Precondition	-
The called functions	-
Input parameter{in}	
direction	the descriptors which users want to init, refer to Table 3-249. Enum <i>enet_dmadirection_enum</i> 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' 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 3-261. 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 3-262. 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 3-263. 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 Table 3-235. Structure enet_descriptors_struct
Input parameter{in}	
checksum	IP frame checksum configuration only one parameter can be selected which is shown as below
ENET_CHECKSUM_DISABLE	checksum insertion disabled
ENET_CHECKSUM_IPV4HEADER	only IP header checksum calculation and insertion are enabled
ENET_CHECKSUM_TCPUDPICMP_SEGMENT	TCP/UDP/ICMP checksum insertion calculated but pseudo-header
ENET_CHECKSUM_TCPUDPICMP_FULL	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_TCPUDPICMP_FULL);
```

enet_enable

The description of enet_enable is shown as below:

Table 3-264. Function enet_enable

Function name	enet_enable
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 3-265. 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 3-266. 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, refer to Table 3-252. Enum enet_macaddress_enum only one parameter can be selected which is shown as below
ENET_MAC_ADDRES S0	set MAC address 0 filter
ENET_MAC_ADDRES S1	set MAC address 1 filter
ENET_MAC_ADDRES S2	set MAC address 2 filter
ENET_MAC_ADDRES S3	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 3-267. Function `enet_mac_address_get`

Function name	<code>enet_mac_address_get</code>
Function prototype	<code>void enet_mac_address_get(enet_macaddress_enum mac_addr, uint8_t paddr[]);</code>
Function descriptions	get MAC address
Precondition	-
The called functions	-
Input parameter{in}	
mac_addr	select which MAC address will be set, refer to Table 3-252. Enum <code>enet_macaddress_enum</code> only one parameter can be selected which is shown as below
<code>ENET_MAC_ADDRES S0</code>	set MAC address 0 filter
<code>ENET_MAC_ADDRES S1</code>	set MAC address 1 filter
<code>ENET_MAC_ADDRES S2</code>	set MAC address 2 filter
<code>ENET_MAC_ADDRES S3</code>	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 3-268. Function `enet_flag_get`

Function name	<code>enet_flag_get</code>
Function prototype	<code>FlagStatus enet_flag_get(enet_flag_enum enet_flag);</code>
Function descriptions	get the ENET MAC/MSR/PTP/DMA status flag
Precondition	-
The called functions	-
Input parameter{in}	
enet_flag	ENET status flag, refer to Table 3-237. Enum <code>enet_flag_enum</code> 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 CT</i>	MSC transmit status flag
<i>ENET_MAC_FLAG_TM ST</i>	time stamp trigger status flag
<i>ENET_PTP_FLAG_TS SCO</i>	timestamp second counter overflow flag
<i>ENET_PTP_FLAG_TT M</i>	target time match flag
<i>ENET_MSC_FLAG_RF CE</i>	received frames CRC error flag
<i>ENET_MSC_FLAG_RF AE</i>	received frames alignment error flag
<i>ENET_MSC_FLAG_RG UF</i>	received good unicast frames flag
<i>ENET_MSC_FLAG_TG FSC</i>	transmitted good frames single collision flag
<i>ENET_MSC_FLAG_TG FMSC</i>	transmitted good frames more single collision flag
<i>ENET_MSC_FLAG_TG F</i>	transmitted good frames flag
<i>ENET_DMA_FLAG_TS</i>	transmit status flag
<i>ENET_DMA_FLAG_TP S</i>	transmit process stopped status flag
<i>ENET_DMA_FLAG_TB U</i>	transmit buffer unavailable status flag
<i>ENET_DMA_FLAG_TJ 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>	receive buffer unavailable status flag

<i>U</i>	
<i>ENET_DMA_FLAG_RPS</i>	receive process stopped status flag
<i>ENET_DMA_FLAG_RWT</i>	receive watchdog timeout status flag
<i>ENET_DMA_FLAG_ET</i>	early transmit status flag
<i>ENET_DMA_FLAG_FBE</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_DMA_ERROR</i>	DMA error flag
<i>ENET_DMA_FLAG_EB_TRANSFER_ERROR</i>	transfer error flag
<i>ENET_DMA_FLAG_EB_ACCESS_ERROR</i>	access error flag
<i>ENET_DMA_FLAG_MSC</i>	MSC status flag
<i>ENET_DMA_FLAG_WUM</i>	WUM status flag
<i>ENET_DMA_FLAG_TST</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 3-269. 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 Table 3-238. Enum enet_flag_clear_enum only one parameter can be selected which is shown as below
ENET_DMA_FLAG_TS_CLR	transmit status flag clear
ENET_DMA_FLAG_TPS_CLR	transmit process stopped status flag clear
ENET_DMA_FLAG_TBU_CLR	transmit buffer unavailable status flag clear
ENET_DMA_FLAG_TJT_CLR	transmit jabber timeout status flag clear
ENET_DMA_FLAG_RO_CLR	receive overflow status flag clear
ENET_DMA_FLAG_TU_CLR	transmit underflow status flag clear
ENET_DMA_FLAG_RS_CLR	receive status flag clear
ENET_DMA_FLAG_RBU_CLR	receive buffer unavailable status flag clear
ENET_DMA_FLAG_RPS_CLR	receive process stopped status flag clear
ENET_DMA_FLAG_RWT_CLR	receive watchdog timeout status flag clear
ENET_DMA_FLAG_ET_CLR	early transmit status flag clear
ENET_DMA_FLAG_FBE_CLR	fatal bus error status flag clear
ENET_DMA_FLAG_ER_CLR	early receive status flag clear
ENET_DMA_FLAG_AICLR	abnormal interrupt summary flag clear
ENET_DMA_FLAG_NICLR	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 3-270. 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, refer to Table 3-239. Enum enet_int_enum only one parameter can be selected which is shown as below
ENET_MAC_INT_WUMIM	WUM interrupt mask
ENET_MAC_INT_TMS TIM	timestamp trigger interrupt mask
ENET_MSC_INT_RFC EIM	received frame CRC error interrupt mask
ENET_MSC_INT_RFA EIM	received frames alignment error interrupt mask
ENET_MSC_INT_RGU FIM	received good unicast frames interrupt mask
ENET_MSC_INT_TGF SCIM	transmitted good frames single collision interrupt mask
ENET_MSC_INT_TGF MSCIM	transmitted good frames more single collision interrupt mask
ENET_MSC_INT_TGFI M	transmitted good frames interrupt mask
ENET_DMA_INT_TIE	transmit interrupt enable
ENET_DMA_INT_TPSI E	transmit process stopped interrupt enable
ENET_DMA_INT_TBUI E	transmit buffer unavailable interrupt enable
ENET_DMA_INT_TJTI E	transmit jabber timeout interrupt enable
ENET_DMA_INT_ROIE	receive overflow interrupt enable
ENET_DMA_INT_TUIE	transmit underflow interrupt enable
ENET_DMA_INT_RIE	receive interrupt enable
ENET_DMA_INT_RBUI E	receive buffer unavailable interrupt enable
ENET_DMA_INT_RPSI E	receive process stopped interrupt enable
ENET_DMA_INT_RWT	receive watchdog timeout interrupt enable

<i>IE</i>	
<i>ENET_DMA_INT_ETIE</i>	early transmit interrupt enable
<i>ENET_DMA_INT_FBEIE</i>	fatal bus error interrupt enable
<i>E</i>	
<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 3-271. 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, refer to Table 3-239. Enum enet_int_enum only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_WUMIM</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</i> <i>M</i>	transmitted good frames interrupt mask
<i>ENET_DMA_INT_TIE</i>	transmit interrupt enable
<i>ENET_DMA_INT_TPSI</i> <i>E</i>	transmit process stopped interrupt enable
<i>ENET_DMA_INT_TBUI</i> <i>E</i>	transmit buffer unavailable interrupt enable
<i>ENET_DMA_INT_TJTI</i> <i>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</i> <i>E</i>	receive buffer unavailable interrupt enable
<i>ENET_DMA_INT_RPSI</i> <i>E</i>	receive process stopped interrupt enable
<i>ENET_DMA_INT_RWT</i> <i>IE</i>	receive watchdog timeout interrupt enable
<i>ENET_DMA_INT_ETIE</i>	early transmit interrupt enable
<i>ENET_DMA_INT_FBEI</i> <i>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 3-272. 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, refer to Table 3-240. Enum enet_int_flag_enum only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_FLG</i> <i>G_WUM</i>	WUM status flag
<i>ENET_MAC_INT_FLG</i> <i>G_MSC</i>	MSC status flag
<i>ENET_MAC_INT_FLG</i> <i>G_MSCR</i>	MSC receive status flag
<i>ENET_MAC_INT_FLG</i> <i>G_MSCT</i>	MSC transmit status flag
<i>ENET_MAC_INT_FLG</i> <i>G_TMST</i>	time stamp trigger status flag
<i>ENET_MSC_INT_FLG</i> <i>G_RFCE</i>	received frames CRC error flag
<i>ENET_MSC_INT_FLG</i> <i>G_RFAE</i>	received frames alignment error flag
<i>ENET_MSC_INT_FLG</i> <i>G_RGUF</i>	received good unicast frames flag
<i>ENET_MSC_INT_FLG</i> <i>G_TGFSC</i>	transmitted good frames single collision flag
<i>ENET_MSC_INT_FLG</i> <i>G_TGFMSC</i>	transmitted good frames more single collision flag
<i>ENET_MSC_INT_FLG</i> <i>G_TGF</i>	transmitted good frames flag
<i>ENET_DMA_INT_FLG</i> <i>G_TS</i>	transmit status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TPS</i>	transmit process stopped status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TBU</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TJT</i>	transmit jabber timeout status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RO</i>	receive overflow status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TU</i>	transmit underflow status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RS</i>	receive status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RBU</i>	receive buffer unavailable status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RPS</i>	receive process stopped status flag

ENET_DMA_INT_FLAG_RWT	receive watchdog timeout status flag
ENET_DMA_INT_FLAG_ET	early transmit status flag
ENET_DMA_INT_FLAG_FBE	fatal bus error status flag
ENET_DMA_INT_FLAG_ER	early receive status flag
ENET_DMA_INT_FLAG_AI	abnormal interrupt summary flag
ENET_DMA_INT_FLAG_NI	normal interrupt summary flag
ENET_DMA_INT_FLAG_MSC	MSC status flag
ENET_DMA_INT_FLAG_WUM	WUM status flag
ENET_DMA_INT_FLAG_TST	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 3-273. 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}	
int_flag_clear	clear ENET interrupt flag, refer to Table 3-241. Enum enet_int_flag_clear_enum only one parameter can be selected which is shown as below
ENET_DMA_INT_FLAG_TS_CLR	transmit status flag

ENET_DMA_INT_FLG G_TPS_CLR	transmit process stopped status flag
ENET_DMA_INT_FLG G_TBU_CLR	transmit buffer unavailable status flag
ENET_DMA_INT_FLG G_TJT_CLR	transmit jabber timeout status flag
ENET_DMA_INT_FLG G_RO_CLR	receive overflow status flag
ENET_DMA_INT_FLG G_TU_CLR	transmit underflow status flag
ENET_DMA_INT_FLG G_RS_CLR	receive status flag
ENET_DMA_INT_FLG G_RBU_CLR	receive buffer unavailable status flag
ENET_DMA_INT_FLG G_RPS_CLR	receive process stopped status flag
ENET_DMA_INT_FLG G_RWT_CLR	receive watchdog timeout status flag
ENET_DMA_INT_FLG G_ET_CLR	early transmit status flag
ENET_DMA_INT_FLG G_FBE_CLR	fatal bus error status flag
ENET_DMA_INT_FLG G_ER_CLR	early receive status flag
ENET_DMA_INT_FLG G_AI_CLR	abnormal interrupt summary flag
ENET_DMA_INT_FLG G_NI_CLR	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 3-274. 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 3-275. 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 3-276. 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 3-277. 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 3-278. 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, refer to Table 3-248. Enum enet_registers_type_enum 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_RGUFCNT
<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_address_filter_enable

The description of enet_address_filter_enable is shown as below:

Table 3-279. 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, refer to Table 3-252. Enum enet_macaddress_enum only one parameter can be selected which is shown as below
<i>ENET_MAC_ADDRES S1</i>	enable MAC address 1 filter
<i>ENET_MAC_ADDRES S2</i>	enable MAC address 2 filter

ENET_MAC_ADDRES S3	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 3-280. 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, refer to Table 3-252. Enum enet_macaddress_enum only one parameter can be selected which is shown as below
ENET_MAC_ADDRES S1	enable MAC address 1 filter
ENET_MAC_ADDRES S2	enable MAC address 2 filter
ENET_MAC_ADDRES S3	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 3-281. Function `enet_address_filter_config`

Function name	<code>enet_address_filter_config</code>
Function prototype	<code>void enet_address_filter_config(enet_macaddress_enum mac_addr, uint32_t addr_mask, uint32_t filter_type);</code>
Function descriptions	configure the MAC address filter
Precondition	-
The called functions	-
Input parameter{in}	
mac_addr	select which MAC address will be enable, refer to Table 3-252. Enum enet_macaddress_enum only one parameter can be selected which is shown as below
<code>ENET_MAC_ADDRES S1</code>	enable MAC address 1 filter
<code>ENET_MAC_ADDRES S2</code>	enable MAC address 2 filter
<code>ENET_MAC_ADDRES S3</code>	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
<code>ENET_ADDRESS_MA SK_BYTE0</code>	mask <code>ENET_MAC_ADDR1L[7:0]</code> bits
<code>ENET_ADDRESS_MA SK_BYTE1</code>	mask <code>ENET_MAC_ADDR1L[15:8]</code> bits
<code>ENET_ADDRESS_MA SK_BYTE2</code>	mask <code>ENET_MAC_ADDR1L[23:16]</code> bits
<code>ENET_ADDRESS_MA SK_BYTE3</code>	mask <code>ENET_MAC_ADDR1L [31:24]</code> bits
<code>ENET_ADDRESS_MA SK_BYTE4</code>	mask <code>ENET_MAC_ADDR1H [7:0]</code> bits
<code>ENET_ADDRESS_MA SK_BYTE5</code>	mask <code>ENET_MAC_ADDR1H [15:8]</code> 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
<code>ENET_ADDRESS_FILT ER_SA</code>	The MAC address is used to compared with the SA field of the received frame
<code>ENET_ADDRESS_FILT ER_DA</code>	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 3-282. 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 3-283. 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, refer to Table 3-250. Enum enet_phydirection_enum
ENET_PHY_WRITE	write data to phy register
ENET_PHY_READ	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 3-284. 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 3-285. 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 3-286. 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_PADCRC_DROP</i>	the function of the MAC strips the Pad/FCS field on received frames
<i>ENET_FORWARD_ERRORSFRAMES</i>	the function that all frame received with error except runt error are forwarded to memory
<i>ENET_FORWARD_UNDERSIZED_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_UNDEERSZ_GOODFRAMES);
```

enet_forward_feature_disable

The description of enet_forward_feature_disable is shown as below:

Table 3-287. 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
ENET_AUTO_PADCRC_DROP	the function of the MAC strips the Pad/FCS field on received frames
ENET_FORWARD_ERROR_FRAMES	the function that all frame received with error except runt error are forwarded to memory
ENET_FORWARD_UNDEERSZ_GOODFRAMES	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_UNDEERSZ_GOODFRAMES);
```

enet_fliter_feature_enable

The description of enet_fliter_feature_enable is shown as below:

Table 3-288. 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_fliter_feature_enable(ENET_SRC_FILTER);
```

enet_fliter_feature_disable

The description of enet_fliter_feature_disable is shown as below:

Table 3-289. Function enet_fliter_feature_disable

Function name	enet_fliter_feature_disable
Function prototype	void enet_fliter_feature_disable(uint32_t feature);
Function descriptions	disable 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:

```
/* disable filter source address function */
```

```
enet_fliter_feature_disable(ENET_SRC_FILTER);
```

enet_pauseframe_generate

The description of enet_pauseframe_generate is shown as below:

Table 3-290. 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 3-291. Function `enet_pauseframe_detect_config`

Function name	<code>enet_pauseframe_detect_config</code>
Function prototype	<code>void enet_pauseframe_detect_config(uint32_t detect);</code>
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
<code>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDDETECT</code>	besides the unique multicast address, MAC can also use the MAC0 address to detecting pause frame
<code>ENET_UNIQUE_PAUSEDDETECT</code>	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_PAUSEDDETECT */
```

```
enet_pauseframe_detect_config(ENET_UNIQUE_PAUSEDDETECT);
```

enet_pauseframe_config

The description of `enet_pauseframe_config` is shown as below:

Table 3-292. Function `enet_pauseframe_config`

Function name	<code>enet_pauseframe_config</code>
Function prototype	<code>void enet_pauseframe_config(uint32_t pausetime, uint32_t pause_threshold);</code>
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
<code>ENET_PAUSETIME_MINUS4</code>	pause time minus 4 slot times
<code>ENET_PAUSETIME_MINUS28</code>	pause time minus 28 slot times

NUS28	
ENET_PAUSETIME_MINUS4 NUS144	pause time minus 144 slot times
ENET_PAUSETIME_MINUS256 NUS256	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 3-293. 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
ENET_DEACTIVE_THRESHOLD_256BYTES	threshold level is 256 bytes
ENET_DEACTIVE_THRESHOLD_512BYTES	threshold level is 512 bytes
ENET_DEACTIVE_THRESHOLD_768BYTES	threshold level is 768 bytes
ENET_DEACTIVE_THRESHOLD_1024BYTES	threshold level is 1024 bytes
ENET_DEACTIVE_THRESHOLD_1280BYTES	threshold level is 1280 bytes
ENET_DEACTIVE_THRESHOLD_1536BYTES	threshold level is 1536 bytes

ENET_DEACTIVE_THRESHOLD_1792BYTES	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
ENET_ACTIVE_THRESHOLD_256BYTES	threshold level is 256 bytes
ENET_ACTIVE_THRESHOLD_512BYTES	threshold level is 512 bytes
ENET_ACTIVE_THRESHOLD_768BYTES	threshold level is 768 bytes
ENET_ACTIVE_THRESHOLD_1024BYTES	threshold level is 1024 bytes
ENET_ACTIVE_THRESHOLD_1280BYTES	threshold level is 1280 bytes
ENET_ACTIVE_THRESHOLD_1536BYTES	threshold level is 1536 bytes
ENET_ACTIVE_THRESHOLD_1792BYTES	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 3-294. 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_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:

```
/* 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 3-295. 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 3-296. 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
ENET_DMA_TX	DMA transmit process
ENET_DMA_RX	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 reval;

reval = enet_dmaprocess_state_get(ENET_DMA_RX);

if(ENET_RX_STATE_SUSPENDED == reval){
    do...
}
```

enet_dmaprocess_resume

The description of enet_dmaprocess_resume is shown as below:

Table 3-297. Function `enet_dmaprocess_resume`

Function name	<code>enet_dmaprocess_resume</code>
Function prototype	<code>void enet_dmaprocess_resume(enet_dmadirection_enum direction);</code>
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
<code>ENET_DMA_TX</code>	DMA transmit process
<code>ENET_DMA_RX</code>	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 3-298. Function `enet_rxprocess_check_recovery`

Function name	<code>enet_rxprocess_check_recovery</code>
Function prototype	<code>void enet_rxprocess_check_recovery(void);</code>
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 3-299. 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;
```

```
reval = enet_txfifo_flush();
```

enet_current_desc_address_get

The description of enet_current_desc_address_get is shown as below:

Table 3-300. 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, refer to Table 3-242. Enum enet_desc_reg_enum only one parameter can be selected which is shown as below
<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

<i>ENET_TX_DESC_TABLE</i>	the start address of the transmit descriptor table
<i>ENET_TX_CURRENT_DESC</i>	the start descriptor address of the current transmit descriptor read by the TxDMA controller
<i>ENET_TX_CURRENT_BUFFER</i>	the current transmit buffer address being read by the TxDMA 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 3-301. Function enet_desc_information_get

Function name	enet_desc_information_get
Function prototype	uint32_t enet_desc_information_get(enet_descriptors_struct *desc, enet_descstate_enum info_get);
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 Table 3-235. Structure enet_descriptors_struct
Input parameter{in}	
info_get	the descriptor information type which is selected, refer to Table 3-253. Enum enet_descstate_enum only one parameter can be selected which is shown as below
<i>RXDESC_BUFFER_1_SIZE</i>	receive buffer 1 size
<i>RXDESC_BUFFER_2_SIZE</i>	receive buffer 2 size
<i>RXDESC_FRAME_LENGTH</i>	the byte length of the received frame that was transferred to the buffer
<i>TXDESC_COLLISION_COUNT</i>	the number of collisions occurred before the frame was transmitted

<code>RXDESC_BUFFER_1_ADDR</code>	the buffer1 address of the Rx frame
<code>TXDESC_BUFFER_1_ADDR</code>	the buffer1 address of the Tx frame
Output parameter{out}	
-	-
Return value	
<code>uint32_t</code>	descriptor information if value is 0xFFFFFFFFU, means the false input parameter

Example:

```
/* get the reception buffer 1 size */
```

```
uint32_t reval;
```

```
enet_descriptors_struct rx_desc;
```

```
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 3-302. Function `enet_missed_frame_counter_get`

Function name	<code>enet_missed_frame_counter_get</code>
Function prototype	<code>void enet_missed_frame_counter_get(uint32_t *rxfifo_drop, uint32_t *rxdma_drop);</code>
Function descriptions	get the number of missed frames during receiving
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
<code>rxfifo_drop</code>	pointer to the number of frames dropped by RxFIFO
Output parameter{out}	
<code>rxdma_drop</code>	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 3-303. 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 Table 3-235. 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
ENET_TDES0_EXD	excessive deferral
ENET_TDES0_VFRM	VLAN frame
ENET_TDES0_ECO	excessive collision
ENET_TDES0_LCO	late collision
ENET_TDES0_NCA	no carrier
ENET_TDES0_LCA	loss of carrier
ENET_TDES0_IPPE	IP payload error
ENET_TDES0_FRMF	frame flushed
ENET_TDES0_JT	jabber timeout
ENET_TDES0_ES	error summary
ENET_TDES0_IPHE	IP header error
ENET_TDES0_TTMSS	transmit timestamp status
ENET_TDES0_TCHM	the second address chained mode
ENET_TDES0_TERM	transmit end of ring mode
ENET_TDES0_TTSEN	transmit timestamp function enable
ENET_TDES0_DPAD	disable adding pad
ENET_TDES0_DCRC	disable CRC
ENET_TDES0_FSG	first segment
ENET_TDES0_LSG	last segment
ENET_TDES0_INTC	interrupt on completion
ENET_TDES0_DAV	DAV bit
When type of parameter desc is RX	

ENET_RDES0_PCERR	payload checksum error
ENET_RDES0_CERR	CRC error
ENET_RDES0_DBERR	dribble bit error
ENET_RDES0_RERR	receive error
ENET_RDES0_RWDT	receive watchdog timeout
ENET_RDES0_FRMT	frame type
ENET_RDES0_LCO	late collision
ENET_RDES0_IPHER R	IP frame header error
ENET_RDES0_LDES	last descriptor
ENET_RDES0_FDES	first descriptor
ENET_RDES0_VTAG	VLAN tag
ENET_RDES0_OERR	overflow error
ENET_RDES0_LERR	length error
ENET_RDES0_SAFF	SA filter fail
ENET_RDES0_DERR	descriptor error
ENET_RDES0_ERRS	error summary
ENET_RDES0_DAFF	destination address filter fail
ENET_RDES0_DAV	descriptor available
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

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

```
FlagStatus reval;
```

```
enet_descriptors_struct p_txdesc;
```

```
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 3-304. Function enet_desc_flag_set

Function name	enet_desc_flag_set
Function prototype	void enet_desc_flag_set(enet_descriptors_struct *desc, uint32_t desc_flag);
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 Table 3-235. 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:

```
/* set VLAN frame bit flag of ENET DMA descriptor */
enet_descriptors_struct p_txdesc;
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 3-305. 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 Table 3-235. 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_TTSN</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_descriptors_struct p_txdesc;
```

```
enet_desc_flag_clear(p_txdesc, ENET_TDES0_TCHM);
```

enet_desc_receive_complete_bit_enable

The description of enet_desc_receive_complete_bit_enable is shown as below:

Table 3-306. Function enet_desc_receive_complete_bit_enable

Function name	enet_desc_receive_complete_bit_enable
Function prototype	void enet_desc_receive_complete_bit_enable(enet_descriptors_struct *desc);
Function descriptions	when receiving the completed, set RS bit in ENET_DMA_STAT register will set
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to get flag, the structure members can refer to Table 3-235. Structure enet_descriptors_struct

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable receive complete bit */
```

```
enet_descriptors_struct p_rxdesc;
```

```
enet_desc_receive_complete_bit_enable(p_rxdesc);
```

enet_desc_receive_complete_bit_disable

The description of enet_desc_receive_complete_bit_disable is shown as below:

Table 3-307. Function enet_desc_receive_complete_bit_disable

Function name	enet_desc_receive_complete_bit_disable
Function prototype	void enet_desc_receive_complete_bit_disable(enet_descriptors_struct *desc);
Function descriptions	when receiving the completed, set RS bit in ENET_DMA_STAT register will not set
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to get flag, the structure members can refer to Table 3-235. Structure enet_descriptors_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable receive complete bit */
```

```
enet_descriptors_struct p_rxdesc;
```

```
enet_desc_receive_complete_bit_disable(p_rxdesc);
```

enet_rxframe_drop

The description of enet_rxframe_drop is shown as below:

Table 3-308. 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 3-309. 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_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:

```
/* 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 3-310. Function `enet_dma_feature_disable`

Function name	<code>enet_dma_feature_disable</code>
Function prototype	<code>void enet_dma_feature_disable(uint32_t feature);</code>
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
<code>ENET_NO_FLUSH_RX FRAME</code>	RxDMA does not flushes frames function
<code>ENET_SECONDFRAM E_OPT</code>	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_ptp_normal_descriptors_chain_init

The description of `enet_ptp_normal_descriptors_chain_init` is shown as below:

Table 3-311. Function `enet_ptp_normal_descriptors_chain_init`

Function name	<code>enet_ptp_normal_descriptors_chain_init</code>
Function prototype	<code>void enet_ptp_normal_descriptors_chain_init(enet_dmadiirection_enum direction, enet_descriptors_struct *desc_ptptab);</code>
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
<code>ENET_DMA_TX</code>	DMA Tx descriptors
<code>ENET_DMA_RX</code>	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 Table 3-235. Structure enet_descriptors_struct

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 3-312. 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_dmadirection_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
ENET_DMA_TX	DMA Tx descriptors
ENET_DMA_RX	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 Table 3-235. Structure enet_descriptors_struct
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 3-313. 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 rx_buffer[500];
uint32_t time_stamp[2];
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 3-314. 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];
```

```
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 3-315. 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 3-316. 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 3-317. 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_WAKEUP_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 3-318. 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
ENET_WUM_MAGIC_PACKET_FRAME	enable a wakeup event due to magic packet reception
ENET_WUM_WAKEUP_FRAME	enable a wakeup event due to wakeup frame reception
ENET_WUM_GLOBAL_UNICAST	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 3-319. 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 3-320. 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
ENET_MSC_COUNTER_STOP_ROLLOVER	counter stop rollover
ENET_MSC_COUNTER_RESET_ON_READ	reset on read
ENET_MSC_COUNTER_FREEZE	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 3-321. 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_COUNTER_STOP_ROLLOVER</i>	counter stop rollover
<i>ENET_MSC_RESET_ON_READ</i>	reset on read
<i>ENET_MSC_COUNTER_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_get

The description of enet_msc_counters_get is shown as below:

Table 3-322. 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_SCCNT</i>	MSC transmitted good frames after a single collision counter
<i>ENET_MSC_TX_MSCCNT</i>	MSC transmitted good frames after more than a single collision counter
<i>ENET_MSC_TX_TGFCNT</i>	MSC transmitted good frames counter
<i>ENET_MSC_RX_RFCECNT</i>	MSC received frames with CRC error counter
<i>ENET_MSC_RX_RFAECNT</i>	MSC received frames with alignment error counter
<i>ENET_MSC_RX_RGUFCNT</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*/
```

```
uint32_t reval;
```

```
reval = enet_msc_counters_get(ENET_MSC_TX_SCCNT);
```

enet_ptp_subsecond_2_nanosecond

The description of enet_ptp_subsecond_2_nanosecond is shown as below:

Table 3-323. Function enet_ptp_subsecond_2_nanosecond

Function name	enet_ptp_subsecond_2_nanosecond
Function prototype	uint32_t enet_ptp_subsecond_2_nanosecond(uint32_t subsecond);
Function descriptions	change subsecond to nanosecond
Precondition	-
The called functions	-
Input parameter{in}	
subsecond	subsecond value (0 – 0x7FFF FFFF)
Output parameter{out}	
-	-
Return value	
uint32_t	the nanosecond value (0 -- 499999999)

Example:

```
/* change subsecond to nanosecond */
```

```
uint32_t reval;
```

```
reval = enet_ptp_subsecond_2_nanosecond(50);
```

enet_ptp_nanosecond_2_subsecond

The description of enet_ptp_nanosecond_2_subsecond is shown as below:

Table 3-324. Function enet_ptp_nanosecond_2_subsecond

Function name	enet_ptp_nanosecond_2_subsecond
Function prototype	uint32_t enet_ptp_nanosecond_2_subsecond(uint32_t nanosecond);
Function descriptions	change nanosecond to subsecond
Precondition	-
The called functions	-
Input parameter{in}	
nanosecond	nanosecond value (0 -- 499999999)

Output parameter{out}	
-	-
Return value	
uint32_t	the subsecond value (0 – 0x7FFF FFFF)

Example:

```
/* change nanosecond to subsecond */
uint32_t reval;
reval = enet_ptp_nanosecond_2_subsecond(50);
```

enet_ptp_feature_enable

The description of enet_ptp_feature_enable is shown as below:

Table 3-325. 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
ENET_RXTX_TIMESTAMP	timestamp function for transmit and receive frames
ENET_PTP_TIMESTAMP_INTERRUPT	timestamp interrupt trigger
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable PTP function for all received frames */
enet_ptp_feature_enable(ENET_RXTX_TIMESTAMP);
```

enet_ptp_feature_disable

The description of enet_ptp_feature_disable is shown as below:

Table 3-326. 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_INTERRUPT</i>	timestamp interrupt trigger
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable PTP function for all received frames */
```

```
enet_ptp_feature_disable(ENET_RXTX_TIMESTAMP);
```

enet_ptp_timestamp_function_config

The description of enet_ptp_timestamp_function_config is shown as below:

Table 3-327. 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
<i>ENET_PTP_ADDEND_UPDATE</i>	addend register update
<i>ENET_PTP_SYSTIME_UPDATE</i>	timestamp update
<i>ENET_PTP_SYSTIME_INIT</i>	timestamp initialize
<i>ENET_PTP_FINEMODE</i>	the system timestamp uses the fine method for updating
<i>ENET_PTP_COARSEMODE</i>	the system timestamp uses the coarse method for updating
Output parameter{out}	

-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* config addend register update function */
```

```
ErrStatus reval;
```

```
reval = 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 3-328. 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 3-329. 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	
-	-

Example:

```
/* 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 3-330. 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
ENET_PTP_ADD_TO_TIME	update value is added to system time
ENET_PTP_SUBSTRACT_FROM_TIME	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 3-331. 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 3-332. 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 Table 3-236. Structure enet_ptp_systime_struct
Return value	
-	-

Example:

```
/* get the current system time */
```

```
enet_ptp_systime_struct systime;
```

```
enet_ptp_system_time_get(&systime);
```

enet_ptp_start

The description of enet_ptp_start is shown as below:

Table 3-333. Function enet_ptp_start

Function name	enet_ptp_start
Function prototype	void enet_ptp_start(int32_t updatemethod, uint32_t init_sec, uint32_t init_subsec, uint32_t carry_cfg, uint32_t accuracy_cfg);
Function descriptions	configure and start PTP timestamp counter
Precondition	-
The called functions	enet_interrupt_disable/ enet_ptp_feature_enable/ enet_ptp_subsecond_increment_config/ enet_ptp_timestamp_addend_config/ enet_ptp_timestamp_function_config/ enet_ptp_flag_get/ enet_ptp_timestamp_update_config
Input parameter{in}	
updatemethod	method for updating
<i>ENET_PTP_FINEMODE</i>	fine correction method
<i>ENET_PTP_COARSEMODE</i>	coarse correction method
Input parameter{in}	
init_sec	second value for initializing system time (0 – 0xFFFF FFFF)
Input parameter{in}	
init_subsec	subsecond value for initializing system time (0 – 0x7FFF FFFF)
Input parameter{in}	
carry_cfg	the value to be added to the accumulator register (in fine method is used) (0 – 0xFFFF FFFF)
Input parameter{in}	
accuracy_cfg	the value to be added to the subsecond value of system time (0 – 0xFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* gconfigure and start PTP timestamp counter*/
```

```
enet_ptp_start(ENET_PTP_FINEMODE, 0, 0, 50, 0);
```

enet_ptp_finecorrection_adjfreq

The description of enet_ptp_finecorrection_adjfreq is shown as below:

Table 3-334. Function enet_ptp_finecorrection_adjfreq

Function name	enet_ptp_finecorrection_adjfreq
Function prototype	void enet_ptp_finecorrection_adjfreq(int32_t carry_cfg);
Function descriptions	adjust frequency in fine method by configure addend register
Precondition	-
The called functions	enet_ptp_timestamp_addend_config/ enet_ptp_timestamp_function_config
Input parameter{in}	
carry_cfg	the value to be added to the accumulator register (0 – 0xFFFF FFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* adjust frequency in fine method by configure addend register */
enet_ptp_finecorrection_adjfreq(50);
```

enet_ptp_coarsecorrection_systime_update

The description of enet_ptp_coarsecorrection_systime_update is shown as below:

Table 3-335. Function enet_ptp_coarsecorrection_systime_update

Function name	enet_ptp_coarsecorrection_systime_update
Function prototype	void enet_ptp_coarsecorrection_systime_update(enet_ptp_systime_struct *systime_struct);
Function descriptions	update system time in coarse method
Precondition	-
The called functions	enet_ptp_nanosecond_2_subsecond/ enet_ptp_timestamp_update_config/ enet_ptp_timestamp_function_config/ enet_ptp_flag_get/ enet_ptp_timestamp_addend_config
Input parameter{in}	
systime_struct	pointer to a enet_ptp_systime_struct structure which contains parameters of PTP system time, the structure members can refer to Table 3-236. Structure enet_ptp_systime_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:


```
/* update system time in coarse method */
```

```
enet_ptp_systime_struct systime_struct;
```

```
systime_struct.second = 0x0000FFFF;
```

```
systime_struct.nanosecond = 0;
```

```
systime_struct.sign = ENET_PTP_TIME_POSITIVE;
```

```
enet_ptp_coarsecorrection_systime_update(&systime_struct);
```

enet_ptp_finecorrection_settime

The description of enet_ptp_finecorrection_settime is shown as below:

Table 3-336. Function enet_ptp_finecorrection_settime

Function name	enet_ptp_finecorrection_settime
Function prototype	void enet_ptp_finecorrection_settime(enet_ptp_systime_struct * systime_struct);
Function descriptions	set system time in fine method
Precondition	-
The called functions	enet_ptp_nanosecond_2_subsecond/ enet_ptp_timestamp_update_config/ enet_ptp_timestamp_function_config/ enet_ptp_flag_get
Input parameter{in}	
systime_struct	pointer to a enet_ptp_systime_struct structure which contains parameters of PTP system time, the structure members can refer to Table 3-236. Structure enet_ptp_systime_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set system time in fine method */
```

```
enet_ptp_systime_struct systime_struct;
```

```
systime_struct.second = 0x0000FFFF;
```

```
systime_struct.nanosecond = 0;
```

```
systime_struct.sign = ENET_PTP_TIME_POSITIVE;
```

```
enet_ptp_finecorrection_settime(&systime_struct);
```

enet_ptp_flag_get

The description of enet_ptp_flag_get is shown as below:

Table 3-337. Function enet_ptp_flag_get

Function name	enet_ptp_flag_get
Function prototype	FlagStatus enet_ptp_flag_get(uint32_t flag);
Function descriptions	get the ptp flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	ptp flag status to be checked
ENET_PTP_ADDEND_UPDATE	addend register update
ENET_PTP_SYSTIME_UPDATE	timestamp update
ENET_PTP_SYSTIME_INIT	timestamp initialize
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the ptp flag status */
```

```
FlagStatus reval;
```

```
reval = enet_ptp_flag_get(ENET_PTP_ADDEND_UPDATE);
```

enet_initpara_reset

The description of enet_initpara_reset is shown as below:

Table 3-338. 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();
```

3.12. 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 [3.12.1](#), the EXMC firmware functions are introduced in chapter [3.12.2](#).

3.12.1. Descriptions of Peripheral registers

EXMC registers are listed in the table shown as below:

Table 3-339. EXMC Registers

Registers	Descriptions
EXMC_SNCTLx (x=0, 1, 2, 3)	SRAM/NOR Flash control registers
EXMC_SNTCFGx (x=0, 1, 2, 3)	SRAM/NOR Flash timing configuration registers
EXMC_SNWTCFGx (x=0, 1, 2, 3)	SRAM/NOR Flash write timing configuration registers
EXMC_NPCTLx (x=1, 2, 3)	NAND Flash/PC card control registers
EXMC_NPINTENx (x=1, 2, 3)	NAND Flash/PC card interrupt enable registers
EXMC_NPCTCFGx (x=1, 2, 3)	NAND Flash/PC card common space timing configuration registers
EXMC_NPATCFGx (x=1, 2, 3)	NAND Flash/PC card attribute space timing configuration registers
EXMC_PIOTCFG3	PC Card I/O space timing configuration register
EXMC_NECCx (x=1, 2)	NAND Flash ECC registers
EXMC_SDCTLx(x=0, 1)	SDRAM control registers
EXMC_SDTCFGx(x=0, 1)	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

Registers	Descriptions
EXMC_SIDH	SPI ID high register

3.12.2. Descriptions of Peripheral functions

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

Table 3-340. EXMC firmware function

Function name	Function description
exmc_norsram_deinit	deinitialize EXMC NOR/SRAM region
exmc_norsram_struct_para_init	initialize exmc_norsram_parameter_struct with the default values
exmc_norsram_init	initialize EXMC NOR/SRAM region
exmc_norsram_enable	enable EXMC NOR/PSRAM bank region
exmc_norsram_disable	disable EXMC NOR/PSRAM bank region
exmc_nand_deinit	deinitialize EXMC NAND bank
exmc_nand_struct_para_init	initialize exmc_nand_parameter_struct with the default values
exmc_nand_init	initialize EXMC NAND bank
exmc_nand_enable	enable EXMC NAND bank
exmc_nand_disable	disable EXMC NAND bank
exmc_nand_ecc_config	enable or disable the EXMC NAND ECC function
exmc_ecc_get	get the EXMC ECC value
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 PC card bank
exmc_pccard_disable	disable PC card bank
exmc_sdram_deinit	deinitialize EXMC SDRAM device
exmc_sdram_struct_para_init	initialize exmc_sdram_init_struct with the default values
exmc_sdram_init	initialize EXMC SDRAM device
exmc_sdram_struct_command_parameter_init	initialize exmc_sdram_command_parameter_struct with the default values
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	configure the write protection function
exmc_sdram_bankstatus_get	get the status of SDRAM device0 or device1
exmc_sdram_readsample_config	configure the delayed sample clock of read data
exmc_sdram_readsample_enable	enable read sample
exmc_sdram_readsample_disable	disable read sample
exmc_sqpsram_deinit	deinitialize EXMC SQPIPSRAM
exmc_sqpsram_struct_para_init	initialize exmc_sqpsram_init_struct with the default values

Function name	Function description
exmc_sqpsram_init	initialize EXMC SQPIPSRAM
exmc_sqpsram_read_command_set	set the read command
exmc_sqpsram_write_command_set	set the write command
exmc_sqpsram_read_id_command_send	send SPI read ID command
exmc_sqpsram_write_cmd_send	send SPI special command which does not have address and data phase
exmc_sqpsram_low_id_get	get the EXMC SPI ID low data
exmc_sqpsram_high_id_get	get the EXMC SPI ID high data
exmc_sqpsram_send_command_state_get	get the bit value of EXMC send write command bit or read ID command
exmc_flag_get	get EXMC flag status
exmc_flag_clear	clear EXMC flag
exmc_interrupt_enable	enable EXMC interrupt
exmc_interrupt_disable	disable EXMC interrupt
exmc_interrupt_flag_get	get EXMC interrupt flag
exmc_interrupt_flag_clear	clear EXMC interrupt flag

Structure exmc_norsram_timing_parameter_struct

Table 3-341. 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_setup_time	configure the data setup time, asynchronous access mode valid
asyn_address_hold_time	configure the address hold time, asynchronous access mode valid
asyn_address_setup_time	configure the data setup time, asynchronous access mode valid

Structure exmc_norsram_parameter_struct

Table 3-342. Structure exmc_norsram_parameter_struct

Member name	Function description
norsram_region	select the region of EXMC NOR/SRAM bank
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 bust mode

Member name	Function description
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. The structure members can refer to Table 3-341. Structure <i>exmc_norsram_timing_parameter_struct</i>
write_timing	timing parameters for write when the extended mode is used. The structure members can refer to Table 3-341. Structure <i>exmc_norsram_timing_parameter_struct</i>

Structure *exmc_nand_pccard_timing_parameter_struct*

Table 3-343. 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 3-344. Structure *exmc_nand_parameter_struct*

Member name	Function description
nand_bank	select the bank of NAND
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. The structure members can refer to Table 3-343. Structure <i>exmc_nand_pccard_timing_parameter_struct</i>
attribute_space_timing	the timing parameters for NAND flash attribute space. The structure members can refer to Table 3-343. Structure <i>exmc_nand_pccard_timing_parameter_struct</i>

Structure `exmc_pccard_parameter_struct`

Table 3-345. Structure `exmc_pccard_parameter_struct`

Member name	Function description
<code>atr_latency</code>	configure the latency of ALE low to RB low
<code>ctr_latency</code>	configure the latency of CLE low to RB low
<code>wait_feature</code>	enable or disable the wait feature
<code>common_space_timing</code>	the timing parameters for NAND flash common space. The structure members can refer to Table 3-343. Structure <code>exmc_nand_pccard_timing_parameter_struct</code>
<code>attribute_space_timing</code>	the timing parameters for NAND flash attribute space. The structure members can refer to Table 3-343. Structure <code>exmc_nand_pccard_timing_parameter_struct</code>
<code>io_space_timing</code>	the timing parameters for NAND flash IO space. The structure members can refer to Table 3-343. Structure <code>exmc_nand_pccard_timing_parameter_struct</code>

Structure `exmc_sdram_timing_parameter_struct`

Table 3-346. Structure `exmc_sdram_timing_parameter_struct`

Member name	Function description
<code>row_to_column_delay</code>	configure the row to column delay
<code>row_precharge_delay</code>	configure the row precharge delay
<code>write_recovery_delay</code>	configure the write recovery delay
<code>auto_refresh_delay</code>	configure the auto refresh delay
<code>row_address_select_delay</code>	configure the row address select delay
<code>exit_selfrefresh_delay</code>	configure the exit self-refresh delay
<code>load_mode_register_delay</code>	configure the load mode register delay

Structure `exmc_sdram_parameter_struct`

Table 3-347. Structure `exmc_sdram_parameter_struct`

Member name	Function description
<code>sdram_device</code>	device of SDRAM
<code>pipeline_read_delay</code>	the delay for reading data after CAS latency in HCLK clock cycles
<code>brust_read_switch</code>	enable or disable the burst read
<code>sdclk_config</code>	the SDCLK memory clock for both SDRAM banks
<code>write_protection</code>	enable or disable SDRAM bank write protection function

Member name	Function description
cas_latency	configure the SDRAM CAS latency
internal_bank_number	the number internal banks
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. The structure members can refer to Table 3-346. Structure exmc_sdram_timing_parameter_struct

Structure exmc_sdram_command_parameter_struct

Table 3-348. 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_sqpsram_parameter_struct

Table 3-349. Structure exmc_sqpsram_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 3-350. 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	select the region of bank0

EXMC_BANK0_NORS RAM_REGIONx(x=0..3)	region x of bank0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize EXMC NOR/SRAM region 0 */
```

```
exmc_norsram_deinit(EXMC_BANK0_NORSRAM_REGION0);
```

exmc_norsram_struct_para_init

The description of exmc_norsram_struct_para_init is shown as below:

Table 3-351. 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 exmc_norsram_parameter_struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
exmc_norsram_init_struct	structure for initialization, the structure members can refer to Table 3-342. Structure exmc_norsram_parameter_struct
Return value	
-	-

Example:

```
/* initialize the structure 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 3-352. 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_struct	structure for initialization, the structure members can refer to Table 3-342 . Structure exmc_norsram_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize EXMC NOR/SRAM region */

exmc_norsram_parameter_struct nor_init_struct;

exmc_norsram_timing_parameter_struct nor_timing_init_struct;

/* configure timing parameter */

nor_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;
nor_timing_init_struct.syn_data_latency = EXMC_DATA_LAT_2_CLK;
nor_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;
nor_timing_init_struct.bus_latency = 1;
nor_timing_init_struct.asyn_data_setup_time = 7;
nor_timing_init_struct.asyn_address_hold_time = 2;
nor_timing_init_struct.asyn_address_setup_time = 5;

/* configure EXMC bus parameters */

nor_init_struct.norsram_region = EXMC_BANK0_NORSRAM_REGION0;
nor_init_struct.write_mode = EXMC_ASYNC_WRITE;
nor_init_struct.extended_mode = DISABLE;
nor_init_struct.asyn_wait = DISABLE;
nor_init_struct.nwait_signal = DISABLE;
nor_init_struct.memory_write = ENABLE;
nor_init_struct.nwait_config = EXMC_NWAIT_CONFIG_BEFORE;
nor_init_struct.wrap_burst_mode = DISABLE;
nor_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;

```

```

nor_init_struct.burst_mode = DISABLE;

nor_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;

nor_init_struct.memory_type = EXMC_MEMORY_TYPE_NOR;

nor_init_struct.address_data_mux = ENABLE;

nor_init_struct.read_write_timing = &nor_timing_init_struct;

nor_init_struct.write_timing = &nor_timing_init_struct;

exmc_norsram_init(&nor_init_struct);

```

exmc_norsram_enable

The description of exmc_norsram_enable is shown as below:

Table 3-353. 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 bank region
Precondition	-
The called functions	-
Input parameter{in}	
exmc_norsram_region	specifies the region of NOR/PSRAM bank
EXMC_BANK0_NORSRAM_REGIONx(x=0..3)	region x of bank0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable region 0 of bank0 */

exmc_norsram_enable(EXMC_BANK0_NORSRAM_REGION0);

```

exmc_norsram_disable

The description of exmc_norsram_disable is shown as below:

Table 3-354. 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 bank region
Precondition	-

The called functions	-
Input parameter{in}	
exmc_norsram_region n	specifies the region of NOR/PSRAM bank
<i>EXMC_BANK0_NORSRAM_REGIONx(x=0..3)</i>	region x of bank0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable region 0 of bank0 */
exmc_norsram_disable(EXMC_BANK0_NORSRAM_REGION0);
```

exmc_nand_deinit

The description of exmc_nand_deinit is shown as below:

Table 3-355. 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	select the bank of NAND
<i>EXMC_BANKx_NAND(x=1,2)</i>	bank of NAND
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize bank1 */
exmc_nand_deinit (EXMC_BANK1_NAND);
```

exmc_nand_struct_para_init

The description of exmc_nand_struct_para_init is shown as below:

Table 3-356. 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 exmc_nand_parameter_struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
exmc_nand_init_struct t	structure for initialization, the structure members can refer to Table 3-344. Structure exmc_nand_parameter_struct
Return value	
-	-

Example:

```
/* initialize the structure nand_init_struct */
exmc_nand_parameter_struct nand_init_struct;
exmc_norsram_struct_para_init(&nor_init_struct);
```

exmc_nand_init

The description of exmc_nand_init is shown as below:

Table 3-357. 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 3-344. Structure exmc_nand_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize EXMC NAND bank */
exmc_nand_parameter_struct nand_init_struct;
exmc_nand_pccard_timing_parameter_struct nand_timing_init_struct;
```

```

nand_timing_init_struct.setuptime = 1;

nand_timing_init_struct.waittime = 3;

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 3-358. 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	specifies the NAND bank
EXMC_BANKx_NAND(x=1,2)	bank of NAND
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable NAND bank */

exmc_nand_enable(EXMC_BANK1_NAND);

```

exmc_nand_disable

The description of exmc_nand_disable is shown as below:

Table 3-359. Function exmc_nand_disable

Function name	exmc_nand_disable
Function prototype	void exmc_nand_disable(uint32_t exmc_nand_bank);
Function descriptions	disable EXMC NAND bank
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_bank	specifies the NAND bank
<i>EXMC_BANKx_NAND(x=1,2)</i>	bank of NAND
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable NAND bank */
```

```
exmc_nand_disable(EXMC_BANK1_NAND);
```

exmc_nand_ecc_config

The description of exmc_nand_ecc_config is shown as below:

Table 3-360. 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	specifies the NAND bank
<i>EXMC_BANKx_NAND(x=1,2)</i>	bank of NAND
Input parameter{in}	
newvalue	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
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 3-361. 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	specifies the NAND bank
EXMC_BANKx_NAND(x=1,2)	bank of NAND
Output parameter{out}	
-	-
Return value	
uint32_t	0-0xFFFFFFFF

Example:

```
/* get the EXMC ECC value */
```

```
uint32_t value;
```

```
value = exmc_ecc_get(EXMC_BANK1_NAND);
```

exmc_pccard_deinit

The description of exmc_pccard_deinit is shown as below:

Table 3-362. 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 3-363. 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 exmc_pccard_parameter_struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
exmc_pccard_init_struct	structure for initialization, the structure members can refer to Table 3-345. Structure exmc_pccard_parameter_struct
Return value	
-	-

Example:

```
/* initialize the structure 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 3-364. 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 3-345 . Structure exmc_pccard_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize EXMC PC card bank */

exmc_pccard_parameter_struct pccard_init_struct;

exmc_nand_pccard_timing_parameter_struct pccard_timing_init_struct;

pccard_timing_init_struct.databus_hiztime = 2;

pccard_timing_init_struct.holdtime = 2;

pccard_timing_init_struct.waittime = 3;

pccard_timing_init_struct.setuptime = 1;

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 = DISABLE;

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 3-365. Function exmc_pccard_enable

Function name	exmc_pccard_enable
Function prototype	void exmc_pccard_enable(void);
Function descriptions	enable PC card bank
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable PC card bank */
```

```
exmc_pccard_enable();
```

exmc_pccard_disable

The description of exmc_pccard_disable is shown as below:

Table 3-366. Function exmc_pccard_disable

Function name	exmc_pccard_disable
Function prototype	void exmc_pccard_disable(void);
Function descriptions	disable PC card bank
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable PC card bank */
```

```
exmc_pccard_disable();
```

exmc_sdram_deinit

The description of exmc_sdram_deinit is shown as below:

Table 3-367. 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	select the SDRAM device
EXMC_SDRAM_DEVICE Ex(x=0, 1)	device x of SDRAM
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* deinitialize EXMC SDRAM device */
exmc_sdram_deinit(EXMC_SDRAM_DEVICE0);
```

exmc_sdram_struct_para_init

The description of exmc_sdram_struct_para_init is shown as below:

Table 3-368. Function exmc_sdram_struct_para_init

Function name	exmc_sdram_struct_para_init
Function prototype	void exmc_sdram_struct_para_init(exmc_sdram_parameter_struct* exmc_sdram_init_struct);
Function descriptions	initialize exmc_sdram_init_struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
exmc_sdram_init_struct	structure for initialization, the structure members can refer to Table 3-347. Structure exmc_sdram_parameter_struct
Return value	
-	-

Example:

```
/* initialize the structure exmc_sdram_init_struct */
exmc_sdram_parameter_struct exmc_sdram_init_struct;
exmc_sdram_struct_para_init(&exmc_sdram_init_struct);
```

exmc_sdram_init

The description of exmc_sdram_init is shown as below:

Table 3-369. 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_sdram_parameter_struct	structure for initialization, the structure members can refer to Table 3-347 . Structure exmc_sdram_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
exmc_sdram_parameter_struct      sdram_init_struct;

sdram_init_struct.sdram_device = sdram_device;

sdram_init_struct.column_address_width = EXMC_SDRAM_COW_ADDRESS_9;
sdram_init_struct.row_address_width = EXMC_SDRAM_ROW_ADDRESS_13;
sdram_init_struct.data_width = EXMC_SDRAM_DATABUS_WIDTH_16B;
sdram_init_struct.internal_bank_number = EXMC_SDRAM_4_INTER_BANK;
sdram_init_struct.cas_latency = EXMC_CAS_LATENCY_3_SDCLK;
sdram_init_struct.write_protection = DISABLE;
sdram_init_struct.sdclk_config = EXMC_SDCLK_PERIODS_2_HCLK;
sdram_init_struct.burst_read_switch = ENABLE;
sdram_init_struct.pipeline_read_delay = EXMC_PIPELINE_DELAY_1_HCLK;
sdram_init_struct.timing = &sdram_timing_init_struct;
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 3-370. 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_init_struct	structure for initialization, the structure members can refer to Table 3-348. 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(&t_exmc_sdram_command_init_struct);
```

exmc_sdram_command_config

The description of exmc_sdram_command_config is shown as below:

Table 3-371. 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_command_init_struct	structure for initialization, the structure members can refer to Table 3-348. Structure exmc_sdram_command_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the SDRAM memory command */
exmc_sdram_command_parameter_struct t_exmc_sdram_command_init_struct;
t_exmc_sdram_command_init_struct.command = EXMC_SDRAM_CLOCK_ENABLE;
t_exmc_sdram_command_init_struct.bank_select = bank_select;
t_exmc_sdram_command_init_struct.auto_refresh_number = EXMC_SDRAM_AUTO_REFRESH_1_SDCLK;
t_exmc_sdram_command_init_struct.mode_register_content = 0U;
exmc_sdram_command_config(&t_exmc_sdram_command_init_struct);
```

exmc_sdram_refresh_count_set

The description of exmc_sdram_refresh_count_set is shown as below:

Table 3-372. 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, 0x0000~0x1FFF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set auto-refresh interval */
exmc_sdram_refresh_count_set(0x01);
```

exmc_sdram_autorefresh_number_set

The description of exmc_sdram_autorefresh_number_set is shown as below:

Table 3-373. 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
<i>EXMC_SDRAM_AUTO_REFRESH_x_SDCLK(x=1..15)</i>	auto-refresh cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the number of successive auto-refresh command */
```

```
exmc_sdram_autorefresh_number_set(EXMC_SDRAM_AUTO_REFLESH_2_SDCLK);
```

exmc_sdram_write_protection_config

The description of exmc_sdram_write_protection_config is shown as below:

Table 3-374. Function exmc_sdram_write_protection_config

Function name	exmc_sdram_write_protection_config
Function prototype	void exmc_sdram_write_protection_config(uint32_t exmc_sdram_device, ControlStatus newvalue);
Function descriptions	configure the write protection function
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sdram_device	specifies the SDRAM device
EXMC_SDRAM_DEVICE Ex(x=0,1)	SDRAM DEVICE
Input parameter{in}	
newvalue	control value
ENABLE	enable function
DISABLE	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the EXMC SDRAM write protection function */
```

```
exmc_sdram_write_protection_config(EXMC_SDRAM_DEVICE0, ENABLE);
```

exmc_sdram_bankstatus_get

The description of exmc_sdram_bankstatus_get is shown as below:

Table 3-375. Function exmc_sdram_bankstatus_get

Function name	exmc_sdram_bankstatus_get
Function prototype	uint32_t exmc_sdram_bankstatus_get(uint32_t exmc_sdram_device);
Function descriptions	get the status of SDRAM device0 or device1
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sdram_device	specifies the SDRAM device
EXMC_SDRAM_DEVICE	SDRAM DEVICE

<i>Ex(x=0,1)</i>	
Output parameter{out}	
-	-
Return value	
uint32_t	the status of SDRAM device
<i>EXMC_SDRAM_DEVICE_0</i> <i>E_NORMAL</i>	normal status
<i>EXMC_SDRAM_DEVICE_0</i> <i>E_SELF_REFRESH</i>	self refresh status
<i>EXMC_SDRAM_DEVICE_0</i> <i>E_POWER_DOWN</i>	power down status

Example:

```
/* get the status of SDRAM device0 */
```

```
uint32_t temp;
```

```
temp = exmc_sdram_bankstatus_get(EXMC_SDRAM_DEVICE0);
```

exmc_sdram_readsample_config

The description of exmc_sdram_readsample_config is shown as below:

Table 3-376. Function exmc_sdram_readsample_config

Function name	exmc_sdram_readsample_config
Function prototype	void exmc_sdram_readsample_config(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
<i>EXMC_SDRAM_X_DELAY_CELL(x=0..15)</i>	SDRAM delay cell
Input parameter{in}	
extra_hclk	sample cycle of read data
<i>EXMC_SDRAM_READ_SAMPLE_X_EXTRAHCLK(x=0,1)</i>	sample cycle
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the delayed sample clock of read data */
```

```
exmc_sdram_readsample_config(EXMC_SDRAM_0_DELAY_CELL,
EXMC_SDRAM_READSAMPLE_0_EXTRAHCLK);
```

exmc_sdram_readsample_enable

The description of exmc_sdram_readsample_enable is shown as below:

Table 3-377. Function exmc_sdram_readsample_enable

Function name	exmc_sdram_readsample_enable
Function prototype	void exmc_sdram_readsample_enable(void);
Function descriptions	enable read sample
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable read sample */
```

```
exmc_sdram_readsample_enable();
```

exmc_sdram_readsample_disable

The description of exmc_sdram_readsample_disable is shown as below:

Table 3-378. Function exmc_sdram_readsample_disable

Function name	exmc_sdram_readsample_disable
Function prototype	void exmc_sdram_readsample_disable(void);
Function descriptions	disable read sample
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable read sample */
```

exmc_sdram_readsample_disable();

exmc_sqpsram_deinit

The description of exmc_sqpsram_deinit is shown as below:

Table 3-379. Function exmc_sqpsram_deinit

Function name	exmc_sqpsram_deinit
Function prototype	void exmc_sqpsram_deinit(void);
Function descriptions	deinitialize EXMC SQPIPSRAM
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize EXMC SQPIPSRAM */
```

```
exmc_sqpsram_deinit();
```

exmc_sqpsram_struct_para_init

The description of exmc_sqpsram_struct_para_init is shown as below:

Table 3-380. Function exmc_sqpsram_struct_para_init

Function name	exmc_sqpsram_struct_para_init
Function prototype	void exmc_sqpsram_struct_para_init(exmc_sqpsram_parameter_struct* exmc_sqpsram_init_struct);
Function descriptions	initialize exmc_sqpsram_init_struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
exmc_sqpsram_init_struct	structure for initialization, the structure members can refer to Table 3-349. Structure exmc_sqpsram_parameter_struct
Return value	
-	-

Example:

```
/* initialize EXMC SQPIPSRAM */
```

```
exmc_sqpsram_parameter_struct exmc_sqpsram_init_struct;
```

```
exmc_sqpsram_init(&exmc_sqpsram_init_struct);
```

exmc_sqpsram_init

The description of exmc_sqpsram_init is shown as below:

Table 3-381. Function exmc_sqpsram_init

Function name	exmc_sqpsram_init
Function prototype	void exmc_sqpsram_init(exmc_sqpsram_parameter_struct* exmc_sqpsram_init_struct);
Function descriptions	initialize EXMC SQPIPSRAM
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sqpsram_parameter_struct	structure for initialization, the structure members can refer to Table 3-349. Structure exmc_sqpsram_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize EXMC SQPIPSRAM */
```

```
exmc_sqpsram_parameter_struct exmc_sqpsram_init_struct;
```

```
exmc_sqpsram_init_struct->sample_polarity = EXMC_SQPIPSRAM_SAMPLE_RISING_EDGE;
```

```
exmc_sqpsram_init_struct->id_length = EXMC_SQPIPSRAM_ID_LENGTH_64B;
```

```
exmc_sqpsram_init_struct->address_bits = EXMC_SQPIPSRAM_ADDR_LENGTH_24B;
```

```
exmc_sqpsram_init_struct->command_bits = EXMC_SQPIPSRAM_COMMAND_LENGTH_8B; exmc_sqpsram_init(&exmc_sqpsram_init_struct);
```

exmc_sqpsram_read_command_set

The description of exmc_sqpsram_read_command_set is shown as below:

Table 3-382. Function exmc_sqpsram_read_command_set

Function name	exmc_sqpsram_read_command_set
Function prototype	void exmc_sqpsram_read_command_set(uint32_t read_command_mode, uint32_t read_wait_cycle, uint32_t read_command_code);
Function descriptions	set the read command
Precondition	-

The called functions	-
Input parameter{in}	
read_command_mode	configure SPI PSRAM read command mode
<i>EXMC_SQPIPSRAM_READ_MODE_DISABLE</i>	not SPI mode
<i>EXMC_SQPIPSRAM_READ_MODE_SPI</i>	SPI mode
<i>EXMC_SQPIPSRAM_READ_MODE_SQPI</i>	SQPI mode
<i>EXMC_SQPIPSRAM_READ_MODE_QPI</i>	QPI mode
Input parameter{in}	
read_wait_cycle	wait cycle number after address phase, 0..15
Input parameter{in}	
read_command_code	read command for AHB read transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the read command */
```

```
exmc_sqipsram_read_command_set(EXMC_SQPIPSRAM_READ_MODE_SPI, 0x01, 0x01);
```

exmc_sqipsram_write_command_set

The description of exmc_sqipsram_write_command_set is shown as below:

Table 3-383. 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_mode	configure SPI PSRAM write command mode
<i>EXMC_SQPIPSRAM_WRITE_MODE_DISABLE</i>	not SPI mode

EXMC_SQPIPSRAM_WRITE_MODE_SPI	SPI mode
EXMC_SQPIPSRAM_WRITE_MODE_SQPI	SQPI mode
EXMC_SQPIPSRAM_WRITE_MODE_QPI	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 write command */
```

```
exmc_sqpsram_write_command_set(EXMC_SQPIPSRAM_READ_MODE_SPI, 0x01, 0x01);
```

exmc_sqpsram_read_id_command_send

The description of exmc_sqpsram_read_id_command_send is shown as below:

Table 3-384. Function exmc_sqpsram_read_id_command_send

Function name	exmc_sqpsram_read_id_command_send
Function prototype	void exmc_sqpsram_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_sqpsram_read_id_command_send();
```

exmc_sqpsram_write_cmd_send

The description of exmc_sqpsram_write_cmd_send is shown as below:

Table 3-385. Function exmc_sqpsram_write_cmd_send

Function name	exmc_sqpsram_write_cmd_send
Function prototype	void exmc_sqpsram_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 */
exmc_sqpsram_write_cmd_send();
```

exmc_sqpsram_low_id_get

The description of exmc_sqpsram_low_id_get is shown as below:

Table 3-386. Function exmc_sqpsram_low_id_get

Function name	exmc_sqpsram_low_id_get
Function prototype	uint32_t exmc_sqpsram_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:

```
/* get the EXMC SPI ID low data */
uint32_t temp;
temp = exmc_sqpsram_low_id_get();
```

exmc_sqpsram_high_id_get

The description of exmc_sqpsram_high_id_get is shown as below:

Table 3-387. Function exmc_sqpsram_high_id_get

Function name	exmc_sqpsram_high_id_get
Function prototype	uint32_t exmc_sqpsram_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:

```
/* get the EXMC SPI ID low data */

uint32_t temp;

temp = exmc_sqpsram_high_id_get();
```

exmc_sqpsram_send_command_state_get

The description of exmc_sqpsram_send_command_state_get is shown as below:

Table 3-388. Function exmc_sqpsram_send_command_state_get

Function name	exmc_sqpsram_send_command_state_get
Function prototype	FlagStatus exmc_sqpsram_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
EXMC_SEND_COMMAND_FLAG_RDID	EXMC_SRCMD_RDID flag bit
EXMC_SEND_COMMAND_FLAG_SC	EXMC_SWCMD_SC flag bit
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the bit value of EXMC send write command bit */
```

```
FlagStatus send_command_flag_status;
```

```
send_command_flag_status
```

=

```
exmc_sqpsram_send_command_state_get(EXMC_SEND_COMMAND_FLAG_SC);
```

exmc_flag_get

The description of exmc_flag_get is shown as below:

Table 3-389. Function exmc_flag_get

Function name	exmc_flag_get
Function prototype	FlagStatus exmc_flag_get(uint32_t bank, uint32_t flag);
Function descriptions	get EXMC flag status
Precondition	-
The called functions	-
Input parameter{in}	
bank	specifies the NAND bank , PC card bank
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCA RD	the PC Card bank
EXMC_SDRAM_DEVIC E0	the SDRAM device0
EXMC_SDRAM_DEVIC E1	the SDRAM device1
Input parameter{in}	
flag	flag
EXMC_NAND_PCCAR D_FLAG_RISE	interrupt rising edge status
EXMC_NAND_PCCAR D_FLAG_LEVEL	interrupt high-level status
EXMC_NAND_PCCAR D_FLAG_FALL	interrupt falling edge status
EXMC_NAND_PCCAR D_FLAG_FIFOE	FIFO empty flag
EXMC_SDRAM_FLAG _REFRESH	refresh error interrupt flag
EXMC_SDRAM_FLAG _NREADY	not ready status
Output parameter{out}	
-	-

Return value	
FlagStatus	SET or RESET

Example:

```
/* check EXMC flag is set or not */
```

```
FlagStatus status;
```

```
status = 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 3-390. Function exmc_flag_clear

Function name	exmc_flag_clear
Function prototype	void exmc_flag_clear(uint32_t bank, uint32_t flag);
Function descriptions	clear EXMC flag
Precondition	-
The called functions	-
Input parameter{in}	
bank	specifies the NAND bank, PC card bank
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCA RD	the PC Card bank
EXMC_SDRAM_DEVIC E0	the SDRAM device0
EXMC_SDRAM_DEVIC E1	the SDRAM device1
Input parameter{in}	
flag	flag
EXMC_NAND_PCCAR D_FLAG_RISE	interrupt rising edge status
EXMC_NAND_PCCAR D_FLAG_LEVEL	interrupt high-level status
EXMC_NAND_PCCAR D_FLAG_FALL	interrupt falling edge status
EXMC_NAND_PCCAR D_FLAG_FIFOE	FIFO empty flag
EXMC_SDRAM_FLAG _REFRESH	refresh error interrupt flag
EXMC_SDRAM_FLAG _NREADY	not ready status
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* clear EXMC flag */
```

```
exmc_flag_clear(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_FLAG_RISE);
```

exmc_interrupt_enable

The description of exmc_interrupt_enable is shown as below:

Table 3-391. Function exmc_interrupt_enable

Function name	exmc_interrupt_enable
Function prototype	void exmc_interrupt_enable(uint32_t bank, uint32_t interrupt_source);
Function descriptions	enable EXMC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
bank	specifies the NAND bank, PC card bank
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCARD	the PC card bank
EXMC_SDRAM_DEVICE0	the SDRAM device0
EXMC_SDRAM_DEVICE1	the SDRAM device1
Input parameter{in}	
interrupt_source	specifies get which interrupt flag
EXMC_NAND_PCCARD_INT_RISE	interrupt source of rising edge
EXMC_NAND_PCCARD_INT_LEVEL	interrupt source of high-level
EXMC_NAND_PCCARD_INT_FALL	interrupt source of falling edge
EXMC_SDRAM_INT_REFRESH	interrupt source of refresh error
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable EXMC interrupt */
```

```
exmc_interrupt_enable(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_RISE);
```

exmc_interrupt_disable

The description of exmc_interrupt_disable is shown as below:

Table 3-392. Function exmc_interrupt_disable

Function name	exmc_interrupt_disable
Function prototype	void exmc_interrupt_disable(uint32_t bank, uint32_t interrupt_source);
Function descriptions	disable EXMC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
bank	specifies the NAND bank, PC card bank
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCARD	the PC card bank
RD	
EXMC_SDRAM_DEVICE0	the SDRAM device0
EXMC_SDRAM_DEVICE1	the SDRAM device1
E1	
Input parameter{in}	
interrupt_source	specifies get which interrupt flag
EXMC_NAND_PCCARD_INT_RISE	interrupt source of rising edge
EXMC_NAND_PCCARD_INT_LEVEL	interrupt source of high-level
EXMC_NAND_PCCARD_INT_FALL	interrupt source of falling edge
EXMC_SDRAM_INT_REFRESH	interrupt source of refresh error
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable EXMC interrupt */
```

```
exmc_interrupt_disable(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_RISE);
```

exmc_interrupt_flag_get

The description of exmc_interrupt_flag_get is shown as below:

Table 3-393. Function exmc_interrupt_flag_get

Function name	exmc_interrupt_flag_get
Function prototype	FlagStatus exmc_interrupt_flag_get(uint32_t bank, uint32_t interrupt_source);
Function descriptions	get EXMC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
bank	specifies the NAND bank , PC card bank
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCA RD	the PC Card bank
EXMC_SDRAM_DEVIC E0	the SDRAM device0
EXMC_SDRAM_DEVIC E1	the SDRAM device1
Input parameter{in}	
interrupt_source	Interrupt flag
EXMC_NAND_PCCAR D_INT_RISE	interrupt source of rising edge
EXMC_NAND_PCCAR D_INT_LEVEL	interrupt source of high-level
EXMC_NAND_PCCAR D_INT_FALL	interrupt source of falling edge
EXMC_SDRAM_INT_F LAG_REFRESH	interrupt source of refresh error
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check EXMC interrupt flag is set or not */
```

```
FlagStatus status;
```

```
status = exmc_interrupt_flag_get(EXMC_BANK1_NAND,  
EXMC_NAND_PCCARD_INT_RISE);
```

exmc_interrupt_flag_clear

The description of exmc_interrupt_flag_clear is shown as below:

Table 3-394. Function exmc_interrupt_flag_clear

Function name	exmc_interrupt_flag_clear
Function prototype	void exmc_interrupt_flag_clear(uint32_t bank, uint32_t interrupt_source);
Function descriptions	clear EXMC flag status
Precondition	-
The called functions	-
Input parameter{in}	
bank	specifies the NAND bank , PC card bank
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCA RD	the PC Card bank
EXMC_SDRAM_DEVIC E0	the SDRAM device0
EXMC_SDRAM_DEVIC E1	the SDRAM device1
Input parameter{in}	
interrupt_source	Interrupt flag
EXMC_NAND_PCCAR D_INT_RISE	interrupt source of rising edge
EXMC_NAND_PCCAR D_INT_LEVEL	interrupt source of high-level
EXMC_NAND_PCCAR D_INT_FALL	interrupt source of falling edge
EXMC_SDRAM_INT_F LAG_REFRESH	interrupt source of refresh error
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EXMC interrupt flag */
```

```
exmc_interrupt_flag_clear(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_RISE);
```

3.13. EXTI

EXTI is the interrupt/event controller in the MCU. It contains up to 20 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are

listed in chapter [3.13.1](#), the EXTI firmware functions are introduced in chapter [3.13.2](#).

3.13.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

Table 3-395. EXTI Registers

Registers	Descriptions
EXTI_INTEN	interrupt enable register
EXTI_EVEN	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

3.13.2. Descriptions of Peripheral functions

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

Table 3-396. EXTI firmware function

Function name	Function description
exti_deinit	deinitialize the EXTI
exti_init	initialize the 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 3-397. 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

Member name	Function description
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

Enum exti_mode_enum

Table 3-398. Enum exti_mode_enum

Member name	Function description
EXTI_INTERRUPT	EXTI interrupt mode
EXTI_EVENT	EXTI event mode

Enum exti_trig_type_enum

Table 3-399. 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 3-400. 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 3-401. 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 3-397. Enum exti_line_enum
Input parameter{in}	
mode	EXTI mode, refer to Table 3-398. Enum exti_mode_enum
Input parameter{in}	
trig_type	interrupt trigger type, refer to Table 3-399. 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 3-402. 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 3-397. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the interrupt from EXTI line 0 */
```

```
exti_interrupt_enable(EXTI_0);
```

exti_interrupt_disable

The description of exti_interrupt_disable is shown as below:

Table 3-403. 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 3-397. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the interrupt from EXTI line 0 */
```

```
exti_interrupt_disable(EXTI_0);
```

exti_event_enable

The description of exti_event_enable is shown as below:

Table 3-404. 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 3-397. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the event from EXTI line 0 */
```

```
exti_event_enable(EXTI_0);
```

exti_event_disable

The description of exti_event_disable is shown as below:

Table 3-405. 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 3-397. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the event 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 3-406. 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 3-397. 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 3-407. 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 from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-397. 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 3-408. 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 3-397. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	

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

Example:

```
/* get EXTI line 0 flag status */
```

```
FlagStatus state = 0;
```

```
state = exti_flag_get(EXTI_0);
```

exti_flag_clear

The description of exti_flag_clear is shown as below:

Table 3-409. 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 3-397. 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 3-410. 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 3-397. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	

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

Example:

```
/* get EXTI line 0 interrupt flag status */
```

```
FlagStatus state = 0;
```

```
state = exti_interrupt_flag_get(EXTI_0);
```

exti_interrupt_flag_clear

The description of exti_interrupt_flag_clear is shown as below:

Table 3-411. 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 3-397. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EXTI line 0 interrupt flag status */
```

```
exti_interrupt_flag_clear(EXTI_0);
```

3.14. FMC

There is flash controller and option byte for GD32F20x series. The FMC registers are listed in chapter [3.14.1](#) the FMC firmware functions are introduced in chapter [3.14.2](#).

3.14.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

Table 3-412. FMC Registers

Registers	Descriptions
FMC_WS	wait state register
FMC_KEY0	unlock key register 0
FMC_OBKEY	option byte unlock key register

Registers	Descriptions
FMC_STAT0	status register 0
FMC_CTL0	control register 0
FMC_ADDR0	address register 0
FMC_OBSTAT	option byte status register
FMC_WP	erase/program protection register
FMC_KEY1	unlock key register 1
FMC_STAT1	status register 1
FMC_CTL1	control register 1
FMC_ADDR1	address register 1
FMC_WSEN	wait state enable register
FMC_PID	product ID register

3.14.2. Descriptions of Peripheral functions

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

Table 3-413. FMC firmware function

Function name	Function description
fmc_wscent_set	set the FMC wait state counter
fmc_unlock	unlock the main FMC operation
fmc_bank0_unlock	unlock the FMC bank0 operation
fmc_bank1_unlock	unlock the FMC bank1 operation
fmc_lock	lock the main FMC operation
fmc_bank0_lock	lock the FMC bank0 operation
fmc_bank1_lock	lock the FMC bank1 operation
fmc_page_erase	erase page
fmc_mass_erase	erase whole chip
fmc_bank0_erase	erase whole bank0
fmc_bank1_erase	erase whole bank1
fmc_word_program	program a word at the corresponding address
fmc_halfword_program	program a half word at the corresponding address
ob_unlock	unlock the option byte operation
ob_lock	lock the option byte operation
ob_erase	erase the option byte
ob_write_protection_enable	enable write protection
ob_security_protection_config	configure the option byte security protection
ob_user_write	program option byte user
ob_data_program	program option bytes data
ob_user_get	get the FMC option bytes user
ob_data_get	get the FMC option bytes data
ob_write_protection_get	get the FMC option byte write protection
ob_spc_get	get option byte security protection code value

Function name	Function description
fmc_flag_get	check flag is set or not
fmc_flag_clear	clear the FMC flag
fmc_interrupt_enable	enable FMC interrupt
fmc_interrupt_disable	disable FMC interrupt
fmc_interrupt_flag_get	get FMC interrupt flag state
fmc_interrupt_flag_clear	clear FMC interrupt flag

Enum fmc_state_enum

Table 3-414. Enum fmc_state_enum

Member name	Function description
FMC_READY	the operation has been completed
FMC_BUSY	the operation is in progress
FMC_PGERR	program error
FMC_WPERR	erase/program protection error
FMC_TOERR	timeout error

Enum fmc_interrupt_enum

Table 3-415. Enum fmc_interrupt_enum

Member name	Function description
FMC_INT_BANK0_END	FMC bank0 end of program interrupt
FMC_INT_BANK0_ERR	FMC bank0 error interrupt
FMC_INT_BANK1_END	FMC bank1 end of program interrupt
FMC_INT_BANK1_ERR	FMC bank1 error interrupt

Enum fmc_flag_enum

Table 3-416. Enum fmc_flag_enum

Member name	Function description
FMC_FLAG_BANK0_BUSY	FMC bank0 busy flag
FMC_FLAG_BANK0_PGERR	FMC bank0 operation error flag
FMC_FLAG_BANK0_WPERR	FMC bank0 erase/program protection error flag
FMC_FLAG_BANK0_END	FMC bank0 end of operation flag
FMC_FLAG_OBER	FMC option bytes read error flag

Member name	Function description
R	
FMC_FLAG_BANK1_BUSY	FMC bank1 busy flag
FMC_FLAG_BANK1_PGERR	FMC bank1 operation error flag
FMC_FLAG_BANK1_WPERR	FMC bank1 erase/program protection error flag
FMC_FLAG_BANK1_END	FMC bank1 end of operation flag

Enum fmc_interrupt_flag_enum

Table 3-417. Enum fmc_interrupt_flag_enum

Member name	Function description
FMC_INT_FLAG_BANK0_PGERR	FMC bank0 operation error interrupt flag
FMC_INT_FLAG_BANK0_WPERR	FMC bank0 erase/program protection error interrupt flag
FMC_INT_FLAG_BANK0_END	FMC bank0 end of operation interrupt flag
FMC_INT_FLAG_BANK1_PGERR	FMC bank1 operation error interrupt flag
FMC_INT_FLAG_BANK1_WPERR	FMC bank1 erase/program protection error interrupt flag
FMC_INT_FLAG_BANK1_END	FMC bank1 end of operation interrupt flag

fmc_wscnt_set

The description of fmc_wscnt_set is shown as below:

Table 3-418. Function fmc_wscnt_set

Function name	fmc_wscnt_set
Function prototype	void fmc_wscnt_set(uint32_t wscnt);
Function descriptions	set the FMC wait state counter
Precondition	-
The called functions	-
Input parameter{in}	
wscnt	wait state counter value
WS_WSCNT_0	FMC 0 wait
WS_WSCNT_1	FMC 1 wait
WS_WSCNT_2	FMC 2 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 3-419. 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_bank0_unlock

The description of fmc_bank0_unlock is shown as below:

Table 3-420. Function fmc_bank0_unlock

Function name	fmc_bank0_unlock
Function prototype	void fmc_bank0_unlock(void);
Function descriptions	unlock the main FMC bank0 operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* unlock the main FMC bank0 operation */
```

```
fmc_bank0_unlock();
```

fmc_bank1_unlock

The description of fmc_bank1_unlock is shown as below:

Table 3-421. Function fmc_bank1_unlock

Function name	fmc_bank1_unlock
Function prototype	void fmc_bank1_unlock(void);
Function descriptions	unlock the main FMC bank1 operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock the main FMC bank1 operation */
```

```
fmc_bank1_unlock();
```

fmc_lock

The description of fmc_lock is shown as below:

Table 3-422. 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_bank0_lock

The description of fmc_bank0_lock is shown as below:

Table 3-423. Function fmc_bank0_lock

Function name	fmc_bank0_lock
Function prototype	void fmc_bank0_lock(void);
Function descriptions	lock the main FMC bank0 operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock the main FMC bank0 operation */

fmc_bank0_lock();
```

fmc_bank1_lock

The description of fmc_bank1_lock is shown as below:

Table 3-424. Function fmc_bank1_lock

Function name	fmc_bank1_lock
Function prototype	void fmc_bank1_lock(void);
Function descriptions	lock the main FMC bank1 operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock the main FMC bank1 operation */
```

```
fmc_bank1_lock();
```

fmc_page_erase

The description of fmc_page_erase is shown as below:

Table 3-425. Function fmc_page_erase

Function name	fmc_page_erase
Function prototype	fmc_state_enum fmc_page_erase(uint32_t page_address);
Function descriptions	erase page
Precondition	fmc_unlock
The called functions	fmc_bank0_ready_wait / fmc_bank1_ready_wait
Input parameter{in}	
page_address	the page address to be erased
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-414. Enum fmc_state_enum .

Example:

```
/* erase page */
```

```
fmc_state_enum state;
```

```
state = fmc_page_erase(0x08004000);
```

fmc_mass_erase

The description of fmc_mass_erase is shown as below:

Table 3-426. 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_bank0_ready_wait / fmc_bank1_ready_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-414. Enum fmc_state_enum .

Example:

```
/* erase whole chip */
```

```
fmc_state_enum state;
```

```
state = fmc_mass_erase();
```

fmc_bank0_erase

The description of fmc_bank0_erase is shown as below:

Table 3-427. Function fmc_bank0_erase

Function name	fmc_bank0_erase
Function prototype	fmc_state_enum fmc_bank0_erase(void);
Function descriptions	erase whole bank0
Precondition	fmc_unlock
The called functions	fmc_bank0_ready_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-414. Enum fmc_state_enum .

Example:

```
/* erase bank0 whole chip */
```

```
fmc_state_enum state;
```

```
state = fmc_bank0_erase();
```

fmc_bank1_erase

The description of fmc_bank1_erase is shown as below:

Table 3-428. Function fmc_bank1_erase

Function name	fmc_bank1_erase
Function prototype	fmc_state_enum fmc_bank1_erase();
Function descriptions	erase whole bank1
Precondition	fmc_unlock
The called functions	fmc_bank1_ready_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-414. Enum fmc_state_enum .

Example:

```
/* erase bank1 whole chip */

fmc_state_enum state;

state = fmc_bank1_erase();
```

fmc_word_program

The description of fmc_word_program is shown as below:

Table 3-429. 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_bank0_ready_wait / fmc_bank1_ready_wait
Input parameter{in}	
address	the address to program
Input parameter{in}	
data	the data to program
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-414. Enum fmc_state_enum .

Example:

```
/* program a word at the corresponding address */

fmc_state_enum state;

state = fmc_word_program ( 0x08004000,0xaabbccdd);
```

fmc_halfword_program

The description of fmc_halfword_program is shown as below:

Table 3-430. 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_bank0_ready_wait / fmc_bank1_ready_wait
Input parameter{in}	
address	the address to program
Input parameter{in}	

data	the data to program
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-414. Enum fmc_state_enum .

Example:

```
/* program a half word at the corresponding address */
```

```
fmc_state_enum state;
```

```
state = fmc_halfword_program(0x08004000,0xaabb);
```

ob_unlock

The description of ob_unlock is shown as below:

Table 3-431. Function ob_unlock

Function name	ob_unlock
Function prototype	void ob_unlock(void);
Function descriptions	unlock the option byte operation
Precondition	fmc_unlock
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 3-432. Function ob_lock

Function name	ob_lock
Function prototype	void ob_lock(void);
Function descriptions	lock the option byte operation
Precondition	fmc_lock
The called functions	-
Input parameter{in}	

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

Example:

```
/* lock the option byte operation */
```

```
ob_lock();
```

ob_erase

The description of ob_erase is shown as below:

Table 3-433. Function ob_erase

Function name	ob_erase
Function prototype	void ob_erase(void);
Function descriptions	erase the option byte
Precondition	ob_unlock
The called functions	fmc_bank0_ready_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* erase the FMC option byte */
```

```
ob_erase();
```

ob_write_protection_enable

The description of ob_write_protection_enable is shown as below:

Table 3-434. Function ob_write_protection_enable

Function name	ob_write_protection_enable
Function prototype	fmc_state_enum ob_write_protection_enable(uint32_t ob_wp);
Function descriptions	enable write protection
Precondition	ob_unlock
The called functions	fmc_bank0_ready_wait
Input parameter{in}	
ob_wp	enable write protection
OB_WPx	write protect specify sector x

<i>OB_WP_ALL</i>	write protect all sector
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-414. Enum fmc_state_enum .

Example:

```
/* enable write protection */

fmc_state_enum state;

state = ob_write_protection_enable(OB_WP7);
```

ob_security_protection_config

The description of ob_security_protection_config is shown as below:

Table 3-435. Function ob_security_protection_config

Function name	ob_security_protection_config
Function prototype	fmc_state_enum ob_security_protection_config (uint8_t ob_spc);
Function descriptions	configure the option byte security protection
Precondition	ob_unlock
The called functions	fmc_bank0_ready_wait
Input parameter{in}	
ob_spc	specify security protection
<i>FMC_NSPC</i>	no security protection
<i>FMC_USPC</i>	under security protection
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-414. Enum fmc_state_enum .

Example:

```
/* enable security protection */

fmc_state_enum state;

state = ob_security_protection_config(FMC_USPC);
```

ob_user_write

The description of ob_user_write is shown as below:

Table 3-436. Function ob_user_write

Function name	ob_user_write
Function prototype	fmc_state_enum ob_user_write(uint8_t ob_fwdgt, uint8_t ob_deepsleep, uint8_t ob_stdby, uint8_t ob_boot);

Function descriptions	program option byte user
Precondition	ob_unlock
The called functions	fmc_bank0_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</i> <i>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_stdby	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
Input parameter{in}	
ob_boot	specifies the option byte boot bank value
<i>OB_BOOT_B0</i>	boot from bank0
<i>OB_BOOT_B1</i>	boot from bank1 or bank0 if bank1 is void
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-414. Enum fmc_state_enum .

Example:

```
/* configure user option byte */
```

```
fmc_state_enum state;
```

```
state = ob_user_write(OB_FWDGT_HW, OB_DEEPSLEEP_RST, OB_STDBY_RST,
OB_BOOT_B1);
```

ob_data_program

The description of ob_data_program is shown as below:

Table 3-437. Function ob_data_program

Function name	ob_data_program
Function prototype	fmc_state_enum ob_data_program(uint32_t address, uint8_t data);
Function descriptions	program option bytes data
Precondition	ob_unlock
The called functions	fmc_bank0_ready_wait
Input parameter{in}	
address	0x1ffff804 / 0x1ffff806

Input parameter{in}	
data	the byte to be programmed
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-414. Enum fmc_state_enum .

Example:

```
/* program option bytes data */

fmc_state_enum state;

state = ob_data_program(0x1fff804, 0x56);
```

ob_user_get

The description of ob_user_get is shown as below:

Table 3-438. Function ob_user_get

Function name	ob_user_get
Function prototype	uint8_t ob_user_get(void);
Function descriptions	get the FMC option bytes user
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint8_t	the FMC user option byte values(0x00 – 0xFF)

Example:

```
/* get the FMC user option byte */

uint8_t user;

user = ob_user_get();
```

ob_data_get

The description of ob_data_get is shown as below:

Table 3-439. Function ob_data_get

Function name	ob_data_get
Function prototype	uint8_t ob_data_get(void);
Function descriptions	get the FMC option bytes data
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint8_t	the FMC data option byte values(0x0 – 0xFF)

Example:

```
/* get the FMC data option byte */
```

```
uint8_t data;
```

```
data = ob_data_get();
```

ob_write_protection_get

The description of ob_write_protection_get is shown as below:

Table 3-440. Function ob_write_protection_get

Function name	ob_write_protection_get
Function prototype	uint32_t ob_write_protection_get(void);
Function descriptions	get the FMC option byte write protection
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the FMC write protection option byte value(0x0 – 0xFFFFFFFF)

Example:

```
/* get the FMC option byte write protection */
```

```
uint32_t wp;
```

```
wp = ob_write_protection_get();
```

ob_spc_get

The description of ob_spc_get is shown as below:

Table 3-441. Function ob_spc_get

Function name	ob_spc_get
Function prototype	FlagStatus ob_spc_get(void);
Function descriptions	get option byte security protection code value

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;
```

```
spc = ob_spc_get();
```

fmc_flag_get

The description of fmc_flag_get is shown as below:

Table 3-442. 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	FMC flag, refer to Table 3-416. Enum fmc_flag_enum.
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get FMC flag */
```

```
FlagStatus flag;
```

```
flag = fmc_flag_get(FMC_FLAG_BANK0_END);
```

fmc_flag_clear

The description of fmc_flag_clear is shown as below:

Table 3-443. Function fmc_flag_clear

Function name	fmc_flag_clear
Function prototype	void fmc_flag_clear(uint32_t flag);

Function descriptions	check flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
flag	FMC flag, refer to Table 3-416. Enum fmc_flag_enum.
<i>FMC_FLAG_BANK0_PGERR</i>	FMC bank0 operation error flag
<i>FMC_FLAG_BANK0_WPERR</i>	FMC bank0 erase/program protection error flag
<i>FMC_FLAG_BANK0_END</i>	FMC bank0 end of operation flag
<i>FMC_FLAG_BANK1_PGERR</i>	FMC bank1 operation error flag
<i>FMC_FLAG_BANK1_WPERR</i>	FMC bank1 erase/program protection error flag
<i>FMC_FLAG_BANK1_END</i>	FMC bank1 end of operation flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear FMC flag */
fmc_flag_clear(FMC_FLAG_BANK0_END);
```

fmc_interrupt_enable

The description of fmc_interrupt_enable is shown as below:

Table 3-444. Function fmc_interrupt_enable

Function name	fmc_interrupt_enable
Function prototype	void fmc_interrupt_enable(fmc_interrupt_enum interrupt);
Function descriptions	enable FMC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	FMC interrupt, refer to Table 3-415. Enum fmc_interrupt_enum.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable FMC interrupt */
```

```
fmc_interrupt_enable(FMC_INT_BANK0_END);
```

fmc_interrupt_disable

The description of fmc_interrupt_disable is shown as below:

Table 3-445. Function fmc_interrupt_disable

Function name	fmc_interrupt_disable
Function prototype	void fmc_interrupt_disable(fmc_interrupt_enum interrupt);
Function descriptions	disable FMC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	FMC interrupt, refer to Table 3-415. Enum fmc_interrupt_enum .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable FMC interrupt */
```

```
fmc_interrupt_disable(FMC_INT_BANK0_END);
```

fmc_interrupt_flag_get

The description of fmc_interrupt_flag_get is shown as below:

Table 3-446. Function fmc_interrupt_flag_get

Function name	fmc_interrupt_flag_get
Function prototype	FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum int_flag);
Function descriptions	get FMC interrupt flag state
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	FMC interrupt flag, refer to Table 3-417. Enum fmc_interrupt_flag_enum .
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get FMC flag */
```


FlagStatus flag;

```
flag = fmc_interrupt_flag_get (FMC_INT_FLAG_BANK0_PGERR);
```

fmc_interrupt_flag_clear

The description of fmc_interrupt_flag_get is shown as below:

Table 3-447. Function fmc_interrupt_flag_clear

Function name	fmc_interrupt_flag_clear
Function prototype	FlagStatus fmc_interrupt_flag_clear (fmc_interrupt_flag_enum int_flag);
Function descriptions	clear FMC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	FMC interrupt flag, refer to Table 3-417. Enum fmc_interrupt_flag_enum .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear FMC flag */
```

```
fmc_interrupt_flag_get(FMC_INT_FLAG_BANK0_PGERR);
```

3.15. 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 [3.15.1](#), the FWDGT firmware functions are introduced in chapter [3.15.2](#).

3.15.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

Table 3-448. FWDGT Registers

Registers	Descriptions
FWDGT_CTL	control register
FWDGT_PSC	prescaler register
FWDGT_RLD	reload register
FWDGT_STAT	status register

3.15.2. Descriptions of Peripheral functions

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

Table 3-449. 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 3-450. 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 3-451. 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 3-452. 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 3-453. 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 3-454. 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 3-455. 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 3-456. 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-
FWDGT_PSC_DIV4	FWDGT prescaler set to 4
FWDGT_PSC_DIV8	FWDGT prescaler set to 8
FWDGT_PSC_DIV16	FWDGT prescaler set to 16
FWDGT_PSC_DIV32	FWDGT prescaler set to 32
FWDGT_PSC_DIV64	FWDGT prescaler set to 64
FWDGT_PSC_DIV128	FWDGT prescaler set to 128
FWDGT_PSC_DIV256	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 3-457. 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
FWDGT_FLAG_PUD	a write operation to FWDGT_PSC register is on going
FWDGT_FLAG_RUD	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);
```

3.16. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.16.1](#), the GPIO firmware functions are introduced in chapter [3.16.2](#).

3.16.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

Table 3-458. GPIO Registers

Registers	Descriptions
GPIOx_CTL0	GPIO port control register 0
GPIOx_CTL1	GPIO port control register 1
GPIOx_ISTAT	GPIO port input status register
GPIOx_OCTL	GPIO port output control register

Registers	Descriptions
GPIOx_BOP	GPIO port bit operate register
GPIOx_BC	GPIO port bit clear register
GPIOx_LOCK	GPIO port configuration lock register
AFIO_EC	AFIO event control register
AFIO_PCF0	AFIO port configuration register 0
AFIO_EXTISS0	EXTI sources selection register 0
AFIO_EXTISS1	EXTI sources selection register 1
AFIO_EXTISS2	EXTI sources selection register 2
AFIO_EXTISS3	EXTI sources selection register 3
AFIO_PCF1	AFIO port configuration register 1
AFIO_PCF2	AFIO port configuration register 2
AFIO_PCF3	AFIO port configuration register 3
AFIO_PCF4	AFIO port configuration register 4
AFIO_PCF5	AFIO port configuration register 5

3.16.2. Descriptions of Peripheral functions

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

Table 3-459. GPIO firmware function

Function name	Function description
gpio_deinit	reset GPIO port
gpio_afio_deinit	reset alternate function I/O(AFIO)
gpio_init	GPIO parameter initialization
gpio_bit_set	set GPIO pin
gpio_bit_reset	reset GPIO pin
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_pin_remap_config	configure GPIO pin remap
gpio_pin_remap1_config	configure GPIO pin remap1
gpio_ethernet_phy_select	select ethernet MII or RMII PHY
gpio_exti_source_select	select GPIO pin exti sources
gpio_event_output_config	configure GPIO pin event output
gpio_event_output_enable	enable GPIO pin event output
gpio_event_output_disable	disable GPIO pin event output
gpio_pin_lock	lock GPIO pin bit

gpio_deinit

The description of gpio_deinit is shown as below:

Table 3-460. 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
<i>GPIOx</i>	GPIOx(x = A, B, C, D, E, F, G, H, I)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset GPIOA */
gpio_deinit(GPIOA);
```

gpio_afio_deinit

The description of gpio_afio_deinit is shown as below:

Table 3-461. Function gpio_afio_deinit

Function name	gpio_afio_deinit
Function prototype	void gpio_afio_deinit(void);
Function descriptions	reset alternate function I/O(AFIO)
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset alternate function */
gpio_afio_deinit();
```


gpio_init

The description of gpio_init is shown as below:

Table 3-462. Function gpio_init

Function name	gpio_init
Function prototype	void gpio_init(uint32_t gpio_periph,uint32_t mode,uint32_t speed,uint32_t pin);
Function descriptions	GPIO parameter initialization
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}	
mode	gpio pin mode
<i>GPIO_MODE_AIN</i>	analog input mode
<i>GPIO_MODE_IN_FLOATING</i>	floating input mode
<i>GPIO_MODE_IPD</i>	pull-down input mode
<i>GPIO_MODE_IPU</i>	pull-up input mode
<i>GPIO_MODE_OUT_OD</i>	GPIO output with open-drain
<i>GPIO_MODE_OUT_PP</i>	GPIO output with push-pull
<i>GPIO_MODE_AF_OD</i>	AFIO output with open-drain
<i>GPIO_MODE_AF_PP</i>	AFIO output with push-pull
Input parameter{in}	
speed	gpio output max speed value
<i>GPIO_OSPEED_10MHZ</i>	output max speed 10MHz
<i>GPIO_OSPEED_2MHZ</i>	output max speed 2MHz
<i>GPIO_OSPEED_50MHZ</i>	output max speed 50MHz
Input parameter{in}	
pin	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:

```
/* configure PA0 as analog input mode */
```

```
gpio_init(GPIOA, GPIO_MODE_AIN, GPIO_OSPEED_50MHZ, GPIO_PIN_0);
```

gpio_bit_set

The description of gpio_bit_set is shown as below:

Table 3-463. Function gpio_bit_set

Function name	gpio_bit_set
Function prototype	void gpio_bit_set(uint32_t gpio_periph, uint32_t pin);
Function descriptions	set GPIO pin
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}	
pin	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 */
```

```
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

gpio_bit_reset

The description of gpio_bit_reset is shown as below:

Table 3-464. 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
<i>GPIOx</i>	GPIOx(x = A, B, C, D, E, F, G, H, I)
Input parameter{in}	
pin	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:

```
/* reset PA0 */
```

```
gpio_bit_reset(GPIOA, GPIO_PIN_0);
```

gpio_bit_write

The description of gpio_bit_write is shown as below:

Table 3-465. 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
<i>GPIOx</i>	GPIOx(x = A, B, C, D, E, F, G, H, I)
Input parameter{in}	
pin	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	all pins
Input parameter{in}	
bit_value	SET or RESET
<i>RESET</i>	reset 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 3-466. 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 to Port A */
```

```
gpio_port_write(GPIOA, 0xA5);
```

gpio_input_bit_get

The description of gpio_input_bit_get is shown as below:

Table 3-467. 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
<i>GPIOx</i>	GPIOx(x = A, B, C, D, E, F, G, H, I)
Input parameter{in}	
pin	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	

FlagStatus	SET / 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 3-468. 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
<i>GPIOx</i>	GPIOx(x = A, B, C, D, E, F, G, H, I)
Output parameter{out}	
-	-
Return value	
uint16_t	0x00-0xFF

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 3-469. 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
<i>GPIOx</i>	GPIOx(x = A, B, C, D, E, F, G, H, I)

Input parameter{in}	
pin	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	
FlagStatus	SET / 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 3-470. 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
<i>GPIOx</i>	GPIOx(x = A, B, C, D, E, F, G, H, I)
Output parameter{out}	
-	-
Return value	
Uint16_t	0x00-0xFF

Example:

```
/* get output value of Port A */
```

```
uint16_t port_state;
```

```
port_state = gpio_output_port_get(GPIOA);
```

gpio_pin_remap_config

The description of gpio_pin_remap_config is shown as below:

Table 3-471. Function gpio_pin_remap_config

Function name	gpio_pin_remap_config
----------------------	-----------------------

Function prototype	void gpio_pin_remap_config(uint32_t gpio_remap, ControlStatus newvalue);
Function descriptions	configure GPIO pin remap
Precondition	-
The called functions	-
Input parameter(in)	
gpio_remap	select the pin to remap
<i>GPIO_SPI0_REMAP</i>	SPI0 remapping
<i>GPIO_I2C0_REMAP</i>	I2C0 remapping
<i>GPIO_USART0_REMAP</i>	USART0 remapping
<i>P</i>	
<i>GPIO_USART1_REMAP</i>	USART1 remapping
<i>P</i>	
<i>GPIO_USART2_PARTIAL_REMAP</i>	USART2 partial remapping
<i>GPIO_USART2_FULL_REMAP</i>	USART2 full remapping
<i>GPIO_TIMER0_PARTIAL_REMAP</i>	TIMER0 partial remapping
<i>GPIO_TIMER0_FULL_REMAP</i>	TIMER0 full remapping
<i>GPIO_TIMER1_PARTIAL_REMAP0</i>	TIMER1 partial remapping
<i>GPIO_TIMER1_PARTIAL_REMAP1</i>	TIMER1 partial remapping
<i>GPIO_TIMER1_FULL_REMAP</i>	TIMER1 full remapping
<i>GPIO_TIMER2_PARTIAL_REMAP</i>	TIMER2 partial remapping
<i>GPIO_TIMER2_FULL_REMAP</i>	TIMER2 full remapping
<i>GPIO_TIMER3_REMAP</i>	TIMER3 remapping
<i>P</i>	
<i>GPIO_CAN0_PARTIAL_REMAP</i>	CAN0 partial remapping
<i>GPIO_CAN0_FULL_REMAP</i>	CAN0 full remapping
<i>GPIO_PD01_REMAP</i>	PD01 remapping
<i>GPIO_TIMER4CH3_IRQ_REMAP</i>	TIMER4 channel3 internal remapping
<i>GPIO_ADC0_ETRGIN_S_REMAP</i>	ADC0 external trigger inserted conversion remapping
<i>GPIO_ADC0_ETRGRG_REMAP</i>	ADC0 external trigger regular conversion remapping

<i>GPIO_ADC1_ETRGINS_REMAP</i>	ADC1 external trigger inserted conversion remapping
<i>GPIO_ADC1_ETRGREG_REMAP</i>	ADC1 external trigger regular conversion remapping
<i>GPIO_ENET_REMAP</i>	ENET remapping
<i>GPIO_CAN1_REMAP</i>	CAN1 remapping
<i>GPIO_SWJ_NONJTRST_REMAP</i>	full SWJ(JTAG-DP + SW-DP), but without NJTRST
<i>GPIO_SWJ_SWDPENABLE_REMAP</i>	JTAG-DP disabled and SW-DP enabled
<i>GPIO_SWJ_DISABLE_REMAP</i>	JTAG-DP disabled and SW-DP disabled
<i>GPIO_SPI2_REMAP</i>	SPI2 remapping
<i>GPIO_TIMER1IT1_REMAP</i>	TIMER1 internal trigger 1 remapping
<i>GPIO_PTP_PPS_REMAP</i>	ethernet PTP PPS remapping
<i>GPIO_TIMER8_REMAP</i>	TIMER8 remapping
<i>GPIO_TIMER9_REMAP</i>	TIMER9 remapping
<i>GPIO_TIMER10_REMAP</i>	TIMER10 remapping
<i>GPIO_TIMER12_REMAP</i>	TIMER12 remapping
<i>GPIO_TIMER13_REMAP</i>	TIMER13 remapping
<i>GPIO_EXMC_NADV_REMAP</i>	EXMC_NADV connect/disconnect
Input parameter{in}	
newvalue	是否使能
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*enable SPI0 remapping*/
```

```
gpio_pin_remap_config(GPIO_SPI0_REMAP, ENABLE);
```


gpio_pin_remap1_config

The description of gpio_pin_remap1_config is shown as below:

Table 3-472. Function gpio_pin_remap1_config

Function name	gpio_pin_remap1_config
Function prototype	void gpio_pin_remap1_config(uint8_t remap_reg, uint32_t remap, ControlStatus newvalue);
Function descriptions	configure GPIO pin remap1
Precondition	-
The called functions	-
Input parameter{in}	
remap_reg	AFIO port configuration register
<i>GPIO_PCF2</i>	AFIO port configuration register 2
<i>GPIO_PCF3</i>	AFIO port configuration register 3
<i>GPIO_PCF4</i>	AFIO port configuration register 4
<i>GPIO_PCF5</i>	AFIO port configuration register 5
Input parameter{in}	
remap	select the pin to remap
<i>GPIO_PCF2_DCI_VSY NC_PG9_REMAP</i>	DCI VSYNC remapped to PG9
<i>GPIO_PCF2_DCI_VSY NC_PI5_REMAP</i>	DCI VSYNC remapped to PI5
<i>GPIO_PCF2_DCI_D0_ PC6_REMAP</i>	DCI D0 remapped to PC6
<i>GPIO_PCF2_DCI_D0_ PH9_REMAP</i>	DCI D0 remapped to PH9
<i>GPIO_PCF2_DCI_D1_ PC7_REMAP</i>	DCI D1 remapped to PC7
<i>GPIO_PCF2_DCI_D1_ PH10_REMAP</i>	DCI D1 remapped to PH10
<i>GPIO_PCF2_DCI_D2_ PE0_REMAP</i>	DCI D2 remapped to PE0
<i>GPIO_PCF2_DCI_D2_ PG10_REMAP</i>	DCI D2 remapped to PG10
<i>GPIO_PCF2_DCI_D2_ PH11_REMAP</i>	DCI D2 remapped to PH11
<i>GPIO_PCF2_DCI_D3_ PE1_REMAP</i>	DCI D3 remapped to PE1
<i>GPIO_PCF2_DCI_D3_ PG11_REMAP</i>	DCI D3 remapped to PG11
<i>GPIO_PCF2_DCI_D3_ PH12_REMAP</i>	DCI D3 remapped to PH12

<i>GPIO_PCF2_DCI_D4_ PE4_REMAP</i>	DCI D4 remapped to PE4
<i>GPIO_PCF2_DCI_D4_ PH14_REMAP</i>	DCI D4 remapped to PH14
<i>GPIO_PCF2_DCI_D5_ PD3_REMAP</i>	DCI D5 remapped to PD3
<i>GPIO_PCF2_DCI_D5_ PI4_REMAP</i>	DCI D5 remapped to PI4
<i>GPIO_PCF2_DCI_D6_ PE5_REMAP</i>	DCI D6 remapped to PE5
<i>GPIO_PCF2_DCI_D6_ PI6_REMAP</i>	DCI D6 remapped to PI6
<i>GPIO_PCF2_DCI_D7_ PE6_REMAP</i>	DCI D7 remapped to PE6
<i>GPIO_PCF2_DCI_D7_ PI7_REMAP</i>	D7 remapped to PI7
<i>GPIO_PCF2_DCI_D8_ PH6_REMAP</i>	D8 remapped to PH6
<i>GPIO_PCF2_DCI_D8_ PI1_REMAP</i>	DCI D8 remapped to PI1
<i>GPIO_PCF2_DCI_D9_ PH7_REMAP</i>	DCI D9 remapped to PH7
<i>GPIO_PCF2_DCI_D9_ PI2_REMAP</i>	DCI D9 remapped to PI2
<i>GPIO_PCF2_DCI_D10_ _PD6_REMAP</i>	DCI D10 remapped to PD6
<i>GPIO_PCF2_DCI_D10_ _PI3_REMAP</i>	DCI D10 remapped to PI3
<i>GPIO_PCF2_DCI_D11_ _PF10_REMAP</i>	DCI D11 remapped to PF10
<i>GPIO_PCF2_DCI_D11_ _PH15_REMAP</i>	DCI D11 remapped to PH15
<i>GPIO_PCF2_DCI_D12_ _PG6_REMAP</i>	DCI D12 remapped to PG6
<i>GPIO_PCF2_DCI_D13_ _PG15_REMAP</i>	DCI D12 remapped to PG15
<i>GPIO_PCF2_DCI_D13_ _PI0_REMAP</i>	DCI D13 remapped to PI0
<i>GPIO_PCF2_DCI_HSY NC_PH8_REMAP</i>	DCI HSYNC to PH8
<i>GPIO_PCF2_PH01_RE MAP</i>	PH0/PH1 remapping
<i>GPIO_PCF3_TLI_B5_P</i>	TLI B5 remapped to PA3

<i>A3_REMAP</i>	
<i>GPIO_PCF3_TLI_VSY NC_PA4_REMAP</i>	TLI VSYNC remapped to PA4
<i>GPIO_PCF3_TLI_G2_ PA6_REMAP</i>	TLI G2 remapped to PA6
<i>GPIO_PCF3_TLI_R6_P A8_REMAP</i>	TLI R6 remapped to PA8
<i>GPIO_PCF3_TLI_R4_P A11_REMAP</i>	TLI R4 remapped to PA11
<i>GPIO_PCF3_TLI_R5_P A12_REMAP</i>	TLI R5 remapped to PA12
<i>GPIO_PCF3_TLI_R3_P B0_REMAP</i>	TLI R3 remapped to PB0
<i>GPIO_PCF3_TLI_R6_P B1_REMAP</i>	TLI R6 remapped to PB1
<i>GPIO_PCF3_TLI_B6_P B8_REMAP</i>	TLI B6 remapped to PB8
<i>GPIO_PCF3_TLI_B7_P B9_REMAP</i>	TLI B7 remapped to PB9
<i>GPIO_PCF3_TLI_G4_ PB10_REMAP</i>	TLI G4 remapped to PB10
<i>GPIO_PCF3_TLI_G5_ PB11_REMAP</i>	TLI G5 remapped to PB11
<i>GPIO_PCF3_TLI_HSY NC_PC6_REMAP</i>	TLI HSYNC remapped to PC6
<i>GPIO_PCF3_TLI_G6_ PC7_REMAP</i>	TLI G6 remapped to PC7
<i>GPIO_PCF3_TLI_R2_P C10_REMAP</i>	TLI R2 remapped to PC10
<i>GPIO_PCF3_TLI_G7_ PD3_REMAP</i>	TLI G7 remapped to PD3
<i>GPIO_PCF3_TLI_B2_P D6_REMAP</i>	TLI B2 remapped to PD6
<i>GPIO_PCF3_TLI_B3_P D10_REMAP</i>	TLI B3 remapped to PD10
<i>GPIO_PCF3_TLI_B0_P E4_REMAP</i>	TLI B0 remapped to PE4
<i>GPIO_PCF3_TLI_G0_ PE5_REMAP</i>	TLI G0 remapped to PE5
<i>GPIO_PCF3_TLI_G1_ PE6_REMAP</i>	TLI G1 remapped to PE6
<i>GPIO_PCF3_TLI_G3_ PE11_REMAP</i>	TLI G3 remapped to PE11

<i>GPIO_PCF3_TLI_B4_P E12_REMAP</i>	TLI B4 remapped to PE12
<i>GPIO_PCF3_TLI_DE_ PE13_REMAP</i>	TLI DE remapped to PE13
<i>GPIO_PCF3_TLI_CLK_ PE14_REMAP</i>	TLI CLK remapped to PE14
<i>GPIO_PCF3_TLI_R7_P E15_REMAP</i>	TLI R7 remapped to PE15
<i>GPIO_PCF3_TLI_DE_ PF10_REMAP</i>	TLI DE remapped to PF10
<i>GPIO_PCF3_TLI_R7_P G6_REMAP</i>	TLI R7 remapped to PG6
<i>GPIO_PCF3_TLI_CLK_ PG7_REMAP</i>	TLI CLK remapped to PG7
<i>GPIO_PCF3_TLI_G3_ PG10_REMAP</i>	TLI G3 remapped to PG10
<i>GPIO_PCF3_TLI_B2_P G10_REMAP</i>	TLI B2 remapped to PG10
<i>GPIO_PCF3_TLI_B3_P G11_REMAP</i>	TLI B3 remapped to PG11
<i>GPIO_PCF4_TLI_B4_P G12_REMAP</i>	B4 remapped to PG12
<i>GPIO_PCF4_TLI_B1_P G12_REMAP</i>	B1 remapped to PG12
<i>GPIO_PCF4_TLI_R0_P H2_REMAP2</i>	R0 remapped to PH2
<i>GPIO_PCF4_TLI_R1_P H3_REMAP</i>	TLI R1 remapped to PH3
<i>GPIO_PCF4_TLI_R2_P H8_REMAP</i>	TLI R2 remapped to PH8
<i>GPIO_PCF4_TLI_R3_P H9_REMAP</i>	TLI R3 remapped to PH9
<i>GPIO_PCF4_TLI_R4_P H10_REMAP</i>	TLI R4 remapped to PH10
<i>GPIO_PCF4_TLI_R5_P H11_REMAP</i>	TLI R5 remapped to PH11
<i>GPIO_PCF4_TLI_R6_P H12_REMAP</i>	TLI R6 remapped to PH12
<i>GPIO_PCF4_TLI_G2_ PH13_REMAP</i>	TLI G2 remapped to PH13
<i>GPIO_PCF4_TLI_G3_ PH14_REMAP</i>	TLI G3 remapped to PH14
<i>GPIO_PCF4_TLI_G4_ PH15_REMAP</i>	TLI G4 remapped to PH15

<i>PH15_REMAP</i>	
<i>GPIO_PCF4_TLI_G5_PIO_REMAP</i>	TLI G5 remapped to PI0
<i>GPIO_PCF4_TLI_G6_P11_REMAP</i>	TLI G6 remapped to PI1
<i>GPIO_PCF4_TLI_G7_P12_REMAP</i>	TLI G7 remapped to PI2
<i>GPIO_PCF4_TLI_B4_P14_REMAP</i>	TLI B4 remapped to PI4
<i>GPIO_PCF4_TLI_B5_P15_REMAP</i>	TLI B5 remapped to PI5
<i>GPIO_PCF4_TLI_B6_P16_REMAP</i>	TLI B6 remapped to PI6
<i>GPIO_PCF4_TLI_B7_P17_REMAP</i>	TLI B7 remapped to PI7
<i>GPIO_PCF4_TLI_VSYNC_PI9_REMAP</i>	TLI VSYNC remapped to PI9
<i>GPIO_PCF4_TLI_HSYNC_PI10_REMAP</i>	TLI HSYNC remapped to PI10
<i>GPIO_PCF4_TLI_R0_PH4_REMAP</i>	TLI R0 remapped to PH4
<i>GPIO_PCF4_TLI_R1_PI3_REMAP</i>	TLI R1 remapped to PI3
<i>GPIO_PCF4_SPI1_SCK_PD3_REMAP</i>	SPI1 SCK remapped to PD3
<i>GPIO_PCF4_SPI2_MOSI_PD6_REMAP</i>	SPI2 MOSI remapped to PD6
<i>GPIO_PCF5_I2C2_REMAP0</i>	I2C2 remapping 0
<i>GPIO_PCF5_I2C2_REMAP1</i>	I2C2 remapping 1
<i>GPIO_PCF5_TIMER1_CH0_REMAP</i>	TIMER1 CH0 remapped to PA5
<i>GPIO_PCF5_TIMER4_REMAP</i>	TIMER4 CH0 remapping
<i>GPIO_PCF5_TIMER7_CHON_REMAP0</i>	TIMER7 CHON remapping 0
<i>GPIO_PCF5_TIMER7_CHON_REMAP1</i>	TIMER7 CHON remapping 1
<i>GPIO_PCF5_TIMER7_CH_REMAP</i>	TIMER7 CH remapping
<i>GPIO_PCF5_I2C1_REMAP0</i>	I2C1 remapping 0

<i>GPIO_PCF5_I2C1_REMAP1</i>	I2C1 remapping 1
<i>GPIO_PCF5_SPI1_NSCK_REMAP0</i>	SPI1 NSS/SCK remapping 0
<i>GPIO_PCF5_SPI1_NSCK_REMAP1</i>	SPI1 NSS/SCK remapping 1
<i>GPIO_PCF5_SPI1_IO_REMAP0</i>	SPI1 MISO/MOSI remapping 0
<i>GPIO_PCF5_SPI1_IO_REMAP1</i>	SPI1 MISO/MOSI remapping 1
<i>GPIO_PCF5_UART3_REMAP</i>	UART3 remapping
<i>GPIO_PCF5_TIMER11_REMAP</i>	TIMER11 remapping
<i>GPIO_PCF5_CAN0_ADD_REMAP</i>	CAN0 addition remapping
<i>GPIO_PCF5_ENET_TXD3_REMAP</i>	ETH_TXD3 remapped to PE2
<i>GPIO_PCF5_PPS_HI_REMAP</i>	ETH_PPS_OUT remapped to PG8
<i>GPIO_PCF5_ENET_TXD01_REMAP</i>	ETH_TX_EN/ETH_TXD0/ETH_TXD1 remapping
<i>GPIO_PCF5_ENET_CRSCOL_REMAP</i>	ETH_MII_CRIS/ETH_MII_COL remapping
<i>GPIO_PCF5_ENET_RX_HI_REMAP</i>	ETH_RXD2/ETH_RXD3/ETH_RX_ER remapping
<i>GPIO_PCF5_UART6_REMAP</i>	UART6 remapping
<i>GPIO_PCF5_USART5_CK_PG7_REMAP</i>	USART5 CK remapped to PG7
<i>GPIO_PCF5_USART5_RTS_PG12_REMAP</i>	USART5 RTS remapped to PG12
<i>GPIO_PCF5_USART5_CTS_PG13_REMAP</i>	USART5 CTS remapped to PG13
<i>GPIO_PCF5_USART5_TX_PG14_REMAP</i>	USART5 TX remapped to PG14
<i>GPIO_PCF5_USART5_RX_PG9_REMAP</i>	USART5 RX remapped to PG9
<i>GPIO_PCF5_EXMC_SDNWE_PC0_REMAP</i>	EXMC SDNWE remapped to PC0
<i>GPIO_PCF5_EXMC_SDCKE0_PC3_REMAP</i>	EXMC SDCKE0 remapped to PC3
<i>GPIO_PCF5_EXMC_SDCKE1_PB5_REMAP</i>	EXMC SDCKE1 remapped to PB5

<i>DCKE1_PB5_REMAP</i>	
<i>GPIO_PCF5_EXMC_S</i> <i>DNE0_PC2_REMAP</i>	EXMC SDNE0 remapped to PC2
<i>GPIO_PCF5_EXMC_S</i> <i>DNE1_PB6_REMAP</i>	EXMC SDNE1 remapped to PB6
Input parameter{in}	
newvalue	是否使能
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*enable GPIO_PCF2 UART3 remapping */
```

```
gpio_pin_remap_config(GPIO_PCF2, GPIO_PCF5_UART3_REMAP, ENABLE);
```

gpio_ethernet_phy_select

The description of gpio_ethernet_phy_select is shown as below:

Table 3-473. Function gpio_ethernet_phy_select

Function name	gpio_ethernet_phy_select
Function prototype	void gpio_ethernet_phy_select(uint32_t gpio_enetssel);
Function descriptions	select ethernet MII or RMII PHY
Precondition	-
The called functions	-
Input parameter{in}	
enet_sel	ethernet MII or RMII PHY selection
<i>GPIO_ENET_PHY_MII</i>	configure ethernet MAC for connection with an MII PHY
<i>GPIO_ENET_PHY_RMII</i>	configure ethernet MAC for connection with an RMII PHY
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ethernet MAC for connection with an RMII PHY */
```

```
gpio_ethernet_phy_select(GPIO_ENET_PHY_RMII);
```

gpio_exti_source_select

The description of gpio_exti_source_select is shown as below:

Table 3-474. Function gpio_exti_source_select

Function name	gpio_exti_source_select
Function prototype	void gpio_exti_source_select(uint8_t gpio_outputport, uint8_t gpio_outputpin);
Function descriptions	select GPIO pin exti sources
Precondition	-
The called functions	-
Input parameter{in}	
gpio_outputport	gpio event output port
GPIO_PORT_SOURCE_GPIOx	output port source(x = A, B, C, D, E, F, G, H, I)
Input parameter{in}	
gpio_outputpin	gpio event output pin
GPIO_PIN_SOURCE_x	pin number(x=0..15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure PA0 as EXTI source */
```

```
gpio_exti_source_select(GPIO_PORT_SOURCE_GPIOA, GPIO_PIN_SOURCE_0);
```

gpio_event_output_config

The description of gpio_event_output_config is shown as below:

Table 3-475. Function gpio_event_output_config

Function name	gpio_event_output_config
Function prototype	void gpio_event_output_config(uint8_t gpio_outputport, uint8_t gpio_outputpin);
Function descriptions	configure GPIO pin event output
Precondition	-
The called functions	-
Input parameter{in}	
gpio_outputport	gpio event output port
GPIO_EVENT_PORT_GPIOx	event output port x(x = A, B, C, D, E)
Input parameter{in}	
gpio_outputpin	gpio event output pin

<code>GPIO_EVENT_PIN_x</code>	pin number(x=0..15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure PA0 as the output of event */
```

```
gpio_event_output_config(GPIO_EVENT_PORT_GPIOA, GPIO_EVENT_PIN_0);
```

gpio_event_output_enable

The description of `gpio_event_output_enable` is shown as below:

Table 3-476. Function `gpio_event_output_enable`

Function name	<code>gpio_event_output_enable</code>
Function prototype	<code>void gpio_event_output_enable(void);</code>
Function descriptions	enable GPIO pin event output
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable GPIO pin event output */
```

```
gpio_event_output_enable(void);
```

gpio_event_output_disable

The description of `gpio_event_output_disable` is shown as below:

Table 3-477. Function `gpio_event_output_disable`

Function name	<code>gpio_event_output_disable</code>
Function prototype	<code>void gpio_event_output_disable(void);</code>
Function descriptions	disable GPIO pin event output
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable GPIO pin event output */
```

```
gpio_event_output_disable(void);
```

gpio_pin_lock

The description of gpio_pin_lock is shown as below:

Table 3-478. 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
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	<i>GPIOx</i> (<i>x</i> = A, B, C, D, E, F, G, H, I)
Input parameter{in}	
pin	GPIO pin
<i>GPIO_PIN_x</i>	<i>GPIO_PIN_x</i> (<i>x</i> =0..15)
<i>GPIO_PIN_ALL</i>	all pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock PA0 */
```

```
gpio_pin_lock(GPIOA, GPIO_PIN_0);
```

3.17. HAU

The HASH Acceleration Unit supports acceleration of SHA-1, SHA-224, SHA-256, MD5 algorithm and the HMAC (keyed-hash message authentication code) algorithm. The HAU registers are listed in chapter [3.17.1](#). the HAU firmware functions are introduced in chapter [3.17.2](#).

3.17.1. Descriptions of Peripheral registers

HAU registers are listed in the table shown as below:

Table 3-479. HAU Registers

Registers	Descriptions
HAU_CTL	control register
HAU_DI	data input register
HAU_CFG	configuration register
HAU_DO0	data output register 0
HAU_DO1	data output register 1
HAU_DO2	data output register 2
HAU_DO3	data output register 3
HAU_DO4	data output register 4
HAU_DO5	data output register 5
HAU_DO6	data output register 6
HAU_DO7	data output register 7
HAU_INTEN	interrupt enable register
HAU_STAT	status and interrupt flag register

3.17.2. Descriptions of Peripheral functions

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

Table 3-480. HAU firmware function

Function name	Function description
hau_deinit	reset the HAU peripheral
hau_init	initialize the HAU peripheral parameters
hau_init_struct_para_init	initialize the struct hau_initpara
hau_reset	reset the HAU processor core
hau_last_word_validbits_num_config	configure the number of valid bits in last word of the message
hau_data_write	write data to the IN FIFO
hau_infifo_words_num_get	return the number of words already written into the IN FIFO
hau_digest_read	read the message digest result
hau_digest_calculation_enable	enable digest calculation
hau_multiple_single_dma_config	configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer or not
hau_dma_enable	enable the HAU DMA interface
hau_dma_disable	disable the HAU DMA interface
hau_hash_sha_1	calculate digest using SHA1 in HASH mode
hau_hmac_sha_1	calculate digest using SHA1 in HMAC mode
hau_hash_sha_224	calculate digest using SHA224 in HASH mode
hau_hmac_sha_224	calculate digest using SHA224 in HMAC mode

Function name	Function description
hau_hash_sha_256	calculate digest using SHA256 in HASH mode
hau_hmac_sha_256	calculate digest using SHA256 in HMAC mode
hau_hash_md5	calculate digest using MD5 in HASH mode
hau_hmac_md5	calculate digest using MD5 in HMAC mode
hau_flag_get	get the HAU flag status
hau_flag_clear	clear the HAU flag status
hau_interrupt_enable	enable the HAU interrupts
hau_interrupt_disable	disable the HAU interrupts
hau_interrupt_flag_get	get the HAU interrupt flag status
hau_interrupt_flag_clear	clear the HAU interrupt flag status

Structure hau_init_parameter_struct

Table 3-481. Structure hau_init_parameter_struct

Member name	Function description
algo	algorithm selection: HAU_ALGO_SHA1, HAU_ALGO_SHA224, HAU_ALGO_SHA256, HAU_ALGO_MD5
mode	HAU mode selection: HAU_MODE_HASH, HAU_MODE_HMAC
datatype	data type mode: HAU_SWAPPING_32BIT, HAU_SWAPPING_16BIT, HAU_SWAPPING_8BIT, HAU_SWAPPING_1BIT
keytype	key length mode: HAU_KEY_SHORTER_64, HAU_KEY_LONGGER_64

Structure hau_digest_parameter_struct

Table 3-482. Structure hau_digest_parameter_struct

Member name	Function description
out[8]	message digest result 0-7

hau_deinit

The description of hau_deinit is shown as below:

Table 3-483. Function hau_deinit

Function name	hau_deinit
Function prototype	void hau_deinit(void);
Function descriptions	reset the HAU peripheral
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* reset the HAU peripheral */
```

```
hau_deinit();
```

hau_init

The description of hau_init is shown as below:

Table 3-484. Function hau_init

Function name	hau_init
Function prototype	void hau_init(hau_init_parameter_struct* initpara);
Function descriptions	initialize the HAU peripheral parameters
Precondition	-
The called functions	-
Input parameter{in}	
initpara	HAU init parameter struct, refer to structure Table 3-481. Structure hau_init_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the HAU peripheral parameters */
```

```
hau_init_parameter_struct initpara;
```

```
hau_initpara.algo = HAU_ALGO_MD5;
```

```
hau_initpara.mode = HAU_MODE_HASH;
```

```
hau_initpara.datatype = HAU_SWAPPING_8BIT;
```

```
hau_initpara.keytype = HAU_KEY_SHORTER_64;
```

```
hau_init(&initpara);
```

hau_init_struct_para_init

The description of hau_init_parameter_init is shown as below:

Table 3-485. Function hau_init_parameter_init

Function name	hau_init_struct_para_init
----------------------	---------------------------

Function prototype	void hau_init_struct_para_init (hau_init_parameter_struct* initpara);
Function descriptions	initialize the struct hau_init_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
initpara	HAU init parameter struct, refer to structure Table 3-481. Structure hau_init_parameter_struct
Return value	
-	-

Example:

```
/* initialize the HAU peripheral parameters */
```

```
hau_init_parameter_struct initpara;
```

```
hau_init_struct_para_init(&initpara);
```

hau_reset

The description of hau_reset is shown as below:

Table 3-486. Function hau_reset

Function name	hau_reset
Function prototype	void hau_reset(void);
Function descriptions	reset the HAU processor core
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the HAU processor core */
```

```
hau_reset();
```

hau_last_word_validbits_num_config

The description of hau_last_word_validbits_num_config is shown as below:

Table 3-487. Function hau_last_word_validbits_num_config

Function name	hau_last_word_validbits_num_config
Function prototype	void hau_last_word_validbits_num_config(uint32_t valid_num);
Function descriptions	configure the number of valid bits in last word of the message
Precondition	-
The called functions	-
Input parameter{in}	
valid_num	number of valid bits in last word of the message(0x00 – 0x1F)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the number of valid bits in last word of the message */
hau_last_word_validbits_num_config(0x10);
```

hau_data_write

The description of hau_data_write is shown as below:

Table 3-488. Function hau_data_write

Function name	hau_data_write
Function prototype	void hau_data_write(uint32_t data);
Function descriptions	write data to the IN FIFO
Precondition	-
The called functions	-
Input parameter{in}	
data	data to write(0x0 – 0xFFFFFFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write data to the IN FIFO */
hau_data_write(0x10);
```

hau_infifo_words_num_get

The description of hau_infifo_words_num_get is shown as below:

Table 3-489. Function hau_infifo_words_num_get

Function name	hau_infifo_words_num_get
----------------------	--------------------------

Function prototype	uint32_t hau_infifo_words_num_get(void);
Function descriptions	return the number of words already written into the IN FIFO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	0x0 – 0xFFFFFFFF

Example:

```

/* return the number of words already written into the IN FIFO */

uint32_t num;

num = hau_infifo_words_num_get();

```

hau_digest_read

The description of hau_digest_read is shown as below:

Table 3-490. Function hau_digest_read

Function name	hau_digest_read
Function prototype	void hau_digest_read(hau_digest_parameter_struct* digestpara);
Function descriptions	read the message digest result
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
digestpara	HAU digest parameter struct, refer to structure Table 3-482. Structure hau_digest_parameter_struct
Return value	
-	-

Example:

```

/* read the message digest result */

hau_digest_parameter_struct digestpara;

hau_digest_read(&digestpara);

```

hau_digest_calculation_enable

The description of hau_digest_calculation_enable is shown as below:

Table 3-491. Function hau_digest_calculation_enable

Function name	hau_digest_calculation_enable
Function prototype	void hau_digest_calculation_enable(void);
Function descriptions	enable digest calculation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable digest calculation */
hau_digest_calculation_enable();
```

hau_multiple_single_dma_config

The description of hau_multiple_single_dma_config is shown as below:

Table 3-492. Function hau_multiple_single_dma_config

Function name	hau_multiple_single_dma_config
Function prototype	void hau_multiple_single_dma_config(uint32_t multi_single);
Function descriptions	configure single or multiple DMA is used, and digest calculation at the end of a DMA transfer or not
Precondition	-
The called functions	-
Input parameter{in}	
multi_single	Multiple or single
<i>SINGLE_DMA_AUTO_DIGEST</i>	message padding and message digest calculation at the end of a DMA transfer
<i>MULTIPLE_DMA_NO_DIGEST</i>	multiple DMA transfers needed and CALEN bit is not automatically set at the end of a DMA transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer or not */
hau_multiple_single_dma_config(SINGLE_DMA_AUTO_DIGEST);
```

hau_dma_enable

The description of hau_dma_enable is shown as below:

Table 3-493. Function hau_dma_enable

Function name	hau_dma_enable
Function prototype	void hau_dma_enable(void);
Function descriptions	enable the HAU DMA interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the HAU DMA interface */
hau_dma_enable();
```

hau_dma_disable

The description of hau_dma_disable is shown as below:

Table 3-494. Function hau_dma_disable

Function name	hau_dma_disable
Function prototype	void hau_dma_disable(void);
Function descriptions	disable the HAU DMA interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the HAU DMA interface */
hau_dma_disable();
```

hau_hash_sha_1

The description of hau_hash_sha_1 is shown as below:

Table 3-495. Function hau_hash_sha_1

Function name	hau_hash_sha_1
Function prototype	ErrStatus hau_hash_sha_1(uint8_t *input, uint32_t in_length, uint8_t output[20]);
Function descriptions	calculate digest using SHA1 in HASH mode
Precondition	-
The called functions	-
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA1 in HASH mode */
```

```
ErrStatus status;
```

```
const uint8_t input[16] = {0x01};
```

```
static uint8_t output[20];
```

```
status = hau_hash_sha_1((uint8_t *)input, 0x10, output);
```

hau_hmac_sha_1

The description of hau_hmac_sha_1 is shown as below:

Table 3-496. Function hau_hmac_sha_1

Function name	hau_hmac_sha_1
Function prototype	ErrStatus hau_hmac_sha_1(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[20]);
Function descriptions	calculate digest using SHA1 in HMAC mode
Precondition	-
The called functions	-
Input parameter{in}	
key	pointer to the key used for HMAC
Input parameter{in}	
keysize	length of the key used for HMAC

Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA1 in HMAC mode */
```

```
ErrStatus status;
```

```
const uint8_t key[16] = {0x01};
```

```
const uint8_t input[16] = {0x01};
```

```
static uint8_t output[20];
```

```
status = hau_hmac_sha_1((uint8_t *)key, 0x10, (uint8_t *)input, 0x10, output);
```

hau_hash_sha_224

The description of hau_hash_sha_224 is shown as below:

Table 3-497. Function hau_hash_sha_224

Function name	hau_hash_sha_224
Function prototype	ErrStatus hau_hash_sha_224(uint8_t *input, uint32_t in_length, uint8_t output[28]);
Function descriptions	calculate digest using SHA224 in HASH mode
Precondition	-
The called functions	-
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA224 in HASH mode */
```

```
ErrStatus status;
```

```
const uint8_t input[16] = {0x01};

static uint8_t output[20];

status = hau_hash_sha_224((uint8_t *)input, 0x10, output);
```

hau_hmac_sha_224

The description of hau_hmac_sha_224 is shown as below:

Table 3-498. Function hau_hmac_sha_224

Function name	hau_hmac_sha_224
Function prototype	ErrStatus hau_hmac_sha_224(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[28]);
Function descriptions	calculate digest using SHA224 in HMAC mode
Precondition	-
The called functions	-
Input parameter{in}	
key	pointer to the key used for HMAC
Input parameter{in}	
keysize	length of the key used for HMAC
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA224 in HMAC mode */

ErrStatus status;

const uint8_t key[16] = {0x01};

const uint8_t input[16] = {0x01};

static uint8_t output[20];

status = hau_hmac_sha_224((uint8_t *)key, 0x10, (uint8_t *)input, 0x10, output);
```

hau_hash_sha_256

The description of hau_hash_sha_256 is shown as below:

Table 3-499. Function hau_hash_sha_256

Function name	hau_hash_sha_256
Function prototype	ErrStatus hau_hash_sha_256(uint8_t *input, uint32_t in_length, uint8_t output[32]);
Function descriptions	calculate digest using SHA256 in HASH mode
Precondition	-
The called functions	-
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA256 in HASH mode */
```

```
ErrStatus status;
```

```
const uint8_t input[16] = {0x01};
```

```
static uint8_t output[20];
```

```
status = hau_hash_sha_256((uint8_t *)input, 0x10, output);
```

hau_hmac_sha_256

The description of hau_hmac_sha_256 is shown as below:

Table 3-500. Function hau_hmac_sha_256

Function name	hau_hmac_sha_256
Function prototype	ErrStatus hau_hmac_sha_256(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[32]);
Function descriptions	calculate digest using SHA256 in HMAC mode
Precondition	-
The called functions	-
Input parameter{in}	
key	pointer to the key used for HMAC
Input parameter{in}	
keysize	length of the key used for HMAC
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer

Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA256 in HMAC mode */
```

```
ErrStatus status;
```

```
const uint8_t key[16] ={0x01};
```

```
const uint8_t input[16] ={0x01};
```

```
static uint8_t output[20];
```

```
status = hau_hmac_sha_256((uint8_t *)key, 0x10, (uint8_t *)input, 0x10, output);
```

hau_hash_md5

The description of hau_hash_md5 is shown as below:

Table 3-501. Function hau_hash_md5

Function name	hau_hash_md5
Function prototype	ErrStatus hau_hash_md5(uint8_t *input, uint32_t in_length, uint8_t output[16]);
Function descriptions	calculate digest using MD5 in HASH mode
Precondition	-
The called functions	-
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using MD5 in HASH mode */
```

```
ErrStatus status;
```

```
const uint8_t input[16] ={0x01};
```

```
static uint8_t output[20];
```

```
status = hau_hash_md5((uint8_t *)input, 0x10, output);
```

hau_hmac_md5

The description of hau_hmac_md5 is shown as below:

Table 3-502. Function hau_hmac_md5

Function name	hau_hmac_md5
Function prototype	ErrStatus hau_hmac_md5(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[16]);
Function descriptions	calculate digest using MD5 in HMAC mode
Precondition	-
The called functions	-
Input parameter{in}	
key	pointer to the key used for HMAC
Input parameter{in}	
keysize	length of the key used for HMAC
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using MD5 in HMAC mode */
ErrStatus status;

const uint8_t key[16] = {0x01};
const uint8_t input[16] = {0x01};
static uint8_t output[20];

status = hau_hmac_md5((uint8_t *)key, 0x10, (uint8_t *)input, 0x10, output);
```

hau_flag_get

The description of hau_flag_get is shown as below:

Table 3-503. Function hau_flag_get

Function name	hau_flag_get
Function prototype	FlagStatus hau_flag_get(uint32_t flag);
Function descriptions	get the HAU flag status
Precondition	-
The called functions	-

Input parameter{in}	
flag	HAU flag status
<i>HAU_FLAG_DATA_INPUT</i>	there is enough space (16 bytes) in the input FIFO
<i>HAU_FLAG_CALCULATION_COMPLETE</i>	digest calculation is completed
<i>HAU_FLAG_DMA</i>	DMA is enabled (DMAE =1) or a transfer is processing
<i>HAU_FLAG_BUSY</i>	data block is in process
<i>HAU_FLAG_INFIFO_NO_EMPTY</i>	the input FIFO is not empty
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the HAU flag status */
```

```
FlagStatus status;
```

```
status = hau_flag_get(HAU_FLAG_DMA);
```

hau_flag_clear

The description of hau_flag_clear is shown as below:

Table 3-504. Function hau_flag_clear

Function name	hau_flag_clear
Function prototype	void hau_flag_clear(uint32_t flag);
Function descriptions	clear the HAU flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	HAU flag status
<i>HAU_FLAG_DATA_INPUT</i>	there is enough space (16 bytes) in the input FIFO
<i>HAU_FLAG_CALCULATION_COMPLETE</i>	digest calculation is completed
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the HAU flag status */
```

```
hau_flag_clear(HAU_FLAG_DATA_INPUT);
```

hau_interrupt_enable

The description of `hau_interrupt_enable` is shown as below:

Table 3-505. Function `hau_interrupt_enable`

Function name	<code>hau_interrupt_enable</code>
Function prototype	<code>void hau_interrupt_enable(uint32_t interrupt);</code>
Function descriptions	enable the HAU interrupts
Precondition	-
The called functions	-
Input parameter{in}	
flag	HAU flag status
<code>HAU_INT_DATA_INPUT</code>	a new block can be entered into the IN buffer
<code>HAU_INT_CALCULATION_COMPLETE</code>	calculation complete
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable hau interrupt */
hau_interrupt_enable(HAU_INT_DATA_INPUT);
```

hau_interrupt_disable

The description of `hau_interrupt_disable` is shown as below:

Table 3-506. Function `hau_interrupt_disable`

Function name	<code>hau_interrupt_disable</code>
Function prototype	<code>void hau_interrupt_disable(uint32_t interrupt);</code>
Function descriptions	disable the HAU interrupts
Precondition	-
The called functions	-
Input parameter{in}	
flag	HAU flag status
<code>HAU_INT_DATA_INPUT</code>	a new block can be entered into the IN buffer
<code>HAU_INT_CALCULATION_COMPLETE</code>	calculation complete
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable hau interrupt */
hau_interrupt_disable(HAU_INT_DATA_INPUT);
```

hau_interrupt_flag_get

The description of hau_interrupt_flag_get is shown as below:

Table 3-507. Function hau_interrupt_flag_get

Function name	hau_interrupt_flag_get
Function prototype	FlagStatus hau_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get the HAU interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	HAU interrupt flag status
HAU_INT_FLAG_DATA_INPUT	there is enough space (16 bytes) in the input FIFO
HAU_INT_FLAG_CALCULATION_COMPLETE	digest calculation is completed
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the HAU interrupt flag status */
FlagStatus status;
status = hau_interrupt_flag_get(HAU_INT_FLAG_DATA_INPUT);
```

hau_interrupt_flag_clear

The description of hau_interrupt_flag_clear is shown as below:

Table 3-508. Function hau_interrupt_flag_clear

Function name	hau_interrupt_flag_clear
Function prototype	void hau_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear the HAU interrupt flag status
Precondition	-

The called functions	-
Input parameter{in}	
int_flag	HAU interrupt flag status
<i>HAU_INT_FLAG_DATA_INPUT</i>	there is enough space (16 bytes) in the input FIFO
<i>HAU_INT_FLAG_CALCULATION_COMPLETE</i>	digest calculation is completed
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the HAU interrupt flag status */
hau_interrupt_flag_clear(HAU_INT_FLAG_DATA_INPUT);
```

3.18. 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 [3.18.1](#), the I2C firmware functions are introduced in chapter [3.18.2](#).

3.18.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

Table 3-509. 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

3.18.2. Descriptions of Peripheral functions

I2C firmware functions are listed in the table shown as below:

Table 3-510. I2C firmware function

Function name	Function description
i2c_deinit	reset I2C
i2c_clock_config	configure I2C clock
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	enable dual-address mode
i2c_dualaddr_disable	disable dual-address mode
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_enable	I2C DMA mode enable
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	software reset I2C
i2c_pec_enable	I2C PEC calculation on or off
i2c_pec_transfer_enable	I2C whether to transfer PEC value
i2c_pec_value_get	packet error checking value
i2c_smbus_issue_alert	I2C issue alert through SMBA pin
i2c_smbus_arp_enable	I2C ARP protocol in SMBus switch
i2c_flag_get	check I2C flag is set or not
i2c_flag_clear	clear I2C flag
i2c_interrupt_enable	enable I2C interrupt
i2c_interrupt_disable	disable I2C interrupt
i2c_interrupt_flag_get	check I2C interrupt flag
i2c_interrupt_flag_clear	clear I2C interrupt flag

i2c_deinit

The description of i2c_deinit is shown as below:

Table 3-511. 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
<i>I2Cx</i>	(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 3-512. Function i2c_clock_config

Function name	i2c_clock_config
Function prototype	void i2c_clock_config(uint32_t i2c_periph, uint32_t clk speed, uint32_t duty cyc);
Function descriptions	I2C clock configure
Precondition	-
The called functions	rcu_clock_freq_get
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
clk speed	i2c clock speed
Input parameter{in}	
duty cyc	duty cycle in fast mode or fast mode plus
<i>I2C_DTCY_2</i>	T_low/T_high=2
<i>I2C_DTCY_16_9</i>	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 3-513. 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>	7bits
<i>I2C_ADDFORMAT_10BITS</i>	10bits
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 3-514. 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
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
type	Device or host
<i>I2C_SMBUS_DEVICE</i>	device
<i>I2C_SMBUS_HOST</i>	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 3-515. 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 3-516. Function i2c_ackpos_config

Function name	i2c_ackpos_config
Function prototype	void i2c_ackpos_config(uint32_t i2c_periph, uint32_t pos);
Function descriptions	configure I2C POAP position
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 3-517. 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	master sends slave address
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 3-518. Function i2c_dualaddr_enable

Function name	i2c_dualaddr_enable
Function prototype	void i2c_dualaddr_enable(uint32_t i2c_periph, uint32_t addr);
Function descriptions	enable dual-address mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
addr	the second address in dual-address mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure i2c second slave address */
```

```
i2c_dualaddr_enable(I2C0, I2C_DUADEN_ENABLE);
```

i2c_dualaddr_disable

The description of i2c_dualaddr_disable is shown as below:

Table 3-519. Function i2c_dualaddr_disable

Function name	i2c_dualaddr_disable
----------------------	----------------------

Function prototype	void i2c_dualaddr_disable(uint32_t i2c_periph);
Function descriptions	disable dual-address mode
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 dual-address*/
```

```
i2c_dualaddr_disable(I2C0);
```

i2c_enable

The description of i2c_enable is shown as below:

Table 3-520. 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
<i>I2Cx</i>	(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 3-521. 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
<i>I2Cx</i>	(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 3-522. 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
<i>I2Cx</i>	(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 3-523. 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
<i>I2Cx</i>	(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 3-524. 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
<i>I2Cx</i>	(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 3-525. 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_enable

The description of i2c_dma_enable is shown as below:

Table 3-526. Function i2c_dma_enable

Function name	i2c_dma_enable
Function prototype	void i2c_dma_enable(uint32_t i2c_periph, uint32_t dmastate);
Function descriptions	enable I2C DMA mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
dmastate	On or off
I2C_DMA_ON	DMA mode enable
I2C_DMA_OFF	DMA mode disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 DMA mode enable */
```

```
i2c_dma_enable(I2C0, I2C_DMA_ON);
```

i2c_dma_last_transfer_config

The description of i2c_dma_last_transfer_config is shown as below:

Table 3-527. Function i2c_dma_last_transfer_config

Function name	i2c_dma_last_transfer_config
Function prototype	void i2c_dma_last_transfer_config(uint32_t i2c_periph, uint32_t dmalast);
Function descriptions	configure whether next DMA EOT is DMA last transfer or not
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_config(I2C0, I2C_DMALST_ON);
```

i2c_stretch_scl_low_config

The description of i2c_stretch_scl_low_config is shown as below:

Table 3-528. 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
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
stretchpara	SCL stretching enable or disable
<i>I2C_SCLSTRETCH_EN</i>	SCL stretching is enabled

<i>ABLE</i>	
<i>I2C_SCLSTRETCH_DISABLE</i>	SCL stretching is disabled
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 3-529. 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
<i>I2Cx</i>	(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 3-530. 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	software reset I2C
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
sreset	under reset or not
I2C_SRESET_SET	I2C is under reset
I2C_SRESET_RESET	I2C is not under reset
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* software reset I2C0 */
```

```
i2c_software_reset_config(I2C0, I2C_SRESET_SET);
```

i2c_pec_enable

The description of i2c_pec_enable is shown as below:

Table 3-531. Function i2c_pec_enable

Function name	i2c_pec_enable
Function prototype	void i2c_pec_enable(uint32_t i2c_periph, uint32_t pecstate);
Function descriptions	I2C PEC calculation on or off
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
pecpara	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_enable(I2C0, I2C_PEC_ENABLE);
```

i2c_pec_transfer_enable

The description of i2c_pec_transfer_enable is shown as below:

Table 3-532. Function i2c_pec_transfer_enable

Function name	i2c_pec_transfer_enable
Function prototype	void i2c_pec_transfer_enable(uint32_t i2c_periph, uint32_t pecpara);
Function descriptions	I2C 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_ENABLE</i>	transfer PEC
<i>I2C_PECTRANS_DISABLE</i>	not transfer PEC
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 transfer PEC */

i2c_pec_transfer_enable(I2C0, I2C_PECTRANS_ENABLE);
```

i2c_pec_value_get

The description of i2c_pec_value_get is shown as below:

Table 3-533. 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}	

i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
uint8_t	0..255

Example:

```
/* I2C0 get packet error checking value */
```

```
uint8_t pec_value;
```

```
pec_value = i2c_pec_value_get(I2C0);
```

i2c_smbus_issue_alert

The description of i2c_smbus_issue_alert is shown as below:

Table 3-534. Function i2c_smbus_issue_alert

Function name	i2c_smbus_issue_alert
Function prototype	void i2c_smbus_issue_alert(uint32_t i2c_periph, uint32_t smbuspara);
Function descriptions	I2C issue alert through SMBA pin
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
smbuspara	issue alert through SMBA pin or not
<i>I2C_SALTSEND_ENABLE</i>	issue alert through SMBA pin
<i>I2C_SALTSEND_DISABLE</i>	not issue alert through SMBA pin
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 issue alert through SMBA pin enable*/
```

```
i2c_smbus_issue_alert(I2C0, I2C_SALTSEND_ENABLE);
```

i2c_smbus_arp_enable

The description of i2c_smbus_arp_enable is shown as below:

Table 3-535. Function `i2c_smbus_arp_enable`

Function name	<code>i2c_smbus_arp_enable</code>
Function prototype	<code>void i2c_smbus_arp_enable(uint32_t i2c_periph, uint32_t arpstate);</code>
Function descriptions	enable or disable I2C ARP protocol in SMBus switch
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
arpstate	ARP protocol in SMBus switch
<i>I2C_ARP_ENABLE</i>	enable ARP
<i>I2C_ARP_DISABLE</i>	disable ARP
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 ARP protocol in SMBus switch */
```

```
i2c_smbus_arp_enable(I2C0, I2C_ARP_ENABLE);
```

`i2c_flag_get`

The description of `i2c_flag_get` is shown as below:

Table 3-536. Function `i2c_flag_get`

Function name	<code>i2c_flag_get</code>
Function prototype	<code>FlagStatus i2c_flag_get(uint32_t i2c_periph, uint32_t flag);</code>
Function descriptions	check I2C flag is set or not
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_OUERR</i>	overflow 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
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 3-537. 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
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 <i>I2C_STAT0</i> and reading <i>I2C_STAT1</i>
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 3-538. Function *i2c_interrupt_enable*

Function name	<i>i2c_interrupt_enable</i>
Function prototype	void <i>i2c_interrupt_enable</i> (uint32_t <i>i2c_periph</i> , <i>i2c_interrupt_enum</i> <i>interrupt</i>);
Function descriptions	enable I2C interrupt
Precondition	-
The called functions	-
Input parameter{in}	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
<i>interrupt</i>	interrupt type
<i>I2C_INT_ERR</i>	error interrupt enable
<i>I2C_INT_EV</i>	event interrupt enable
<i>I2C_INT_BUF</i>	buffer interrupt enable
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 3-539. Function i2c_interrupt_disable

Function name	i2c_interrupt_disable
Function prototype	void i2c_interrupt_disable(uint32_t i2c_periph, i2c_interrupt_enum 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
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 3-540. Function i2c_interrupt_flag_get

Function name	i2c_interrupt_flag_get
Function prototype	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
Function descriptions	check I2C interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
int_flag	interrupt flag

<i>I2C_INT_FLAG_SBSE ND</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_PECE 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
Output parameter{out}	
-	-
Return value	
-	-

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 3-541. Function i2c_interrupt_flag_clear

Function name	i2c_interrupt_flag_clear
Function prototype	void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
Function descriptions	clear I2C interrupt flag

Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
int_flag	interrupt flag
<i>I2C_INT_FLAG_ADDS</i> <i>END</i>	address is sent in master mode or received and matches in slave mode 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</i> <i>RB</i>	arbitration lost in master mode interrupt flag
<i>I2C_INT_FLAG_AERR</i>	acknowledge error interrupt flag
<i>I2C_INT_FLAG_OUER</i> <i>R</i>	over-run or under-run situation occurs in slave mode interrupt flag
<i>I2C_INT_FLAG_PECER</i> <i>RR</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBT</i> <i>O</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBA</i> <i>LT</i>	SMBus alert status interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the acknowledge error interrupt flag */
```

```
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_AERR);
```

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 3-542. NVIC Registers

Registers	Descriptions
ISER ⁽¹⁾	interrupt Set Enable Register

Registers	Descriptions
ICER ⁽¹⁾	interrupt Clear Enable Register
ISPR ⁽¹⁾	interrupt Set Pending Register
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_cm3.h file

2. refer to the structure SCB_Type, is defined in the core_cm3.h file

Table 3-543. 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_cm3.h file

3.19.2. Descriptions of Peripheral functions

Enum IRQn_Type

Table 3-544. IRQn_Type

Member name	Function description
WWDGT_IRQn	WWDGT interrupt
LVD_IRQn	LVD from EXTI line interrupt
TAMPER_IRQn	tamper interrupt
RTC_IRQn	RTC global interrupt
FMC_IRQn	FMC global interrupt
RCU_IRQn	RCU global interrupt
EXTI0_IRQn	EXTI line0 interrupt
EXTI1_IRQn	EXTI line1 interrupt
EXTI2_IRQn	EXTI line2 interrupt
EXTI3_IRQn	EXTI line3 interrupt
EXTI4_IRQn	EXTI line4 interrupt
DMA0_Channel0_IRQn	DMA0 channel 0 global interrupt
DMA0_Channel1_IRQn	DMA0 channel 1 global interrupt
DMA0_Channel2_IRQn	DMA0 channel 2 global interrupt
DMA0_Channel3_IRQn	DMA0 channel 3 global interrupt
DMA0_Channel4_IRQn	DMA0 channel 4 global interrupt
DMA0_Channel5_IRQn	DMA0 channel 5 global interrupt
DMA0_Channel6_IRQn	DMA0 channel 6 global interrupt
ADC0_1_IRQn	ADC0 and ADC1 global interrupts
CAN0_TX_IRQn	CAN0 TX interrupt
CAN0_RX0_IRQn	CAN0 RX0 interrupt
CAN0_RX1_IRQn	CAN0 RX1 interrupt
CAN0_EWMC_IRQn	CAN0 EWMC interrupt
EXTI5_9_IRQn	EXTI[9:5] interrupts
TIMER0_BRK_TIMER8_IRQn	TIMER0 break interrupt and TIMER8 global interrupt
TIMER0_UP_TIMER9_IRQn	TIMER0 update Interrupt and TIMER9 global interrupt
TIMER0_TRG_CMT_TIMER10_IRQn	TIMER0 trigger and commutation interrupt and TIMER10 global interrupt
TIMER0_Channel_IRQn	TIMER0 channel capture compare interrupt
TIMER1_IRQn	TIMER1 global interrupt
TIMER2_IRQn	TIMER2 global interrupt
TIMER3_IRQn	TIMER3 global 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 global interrupt
SPI1_IRQn	SPI1 global interrupt
USART0_IRQn	USART0 global interrupt
USART1_IRQn	USART1 global interrupt
USART2_IRQn	USART2 global interrupt
EXTI10_15_IRQn	EXTI[15:10] interrupts
RTC_Alarm_IRQn	RTC alarm from EXTI line interrupt
USBFS_WKUP_IRQn	USBFS wakeUp from EXTI line interrupt
TIMER7_BRK_TIMER11_IRQn	TIMER7 break interrupt and TIMER11 global interrupt
TIMER7_UP_TIMER12_IRQn	TIMER7 update interrupt and TIMER12 global interrupt
TIMER7_TRG_CMT_TIMER13_IRQn	TIMER7 Channel Capture Compare Interrupt
TIMER7_Channel_IRQn	TIMER7 Channel Capture Compare Interrupt
ADC2_IRQn	ADC2 global 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_IRQn	TIMER5 global interrupt
TIMER6_IRQn	TIMER6 global interrupt
DMA1_Channel0_IRQn	DMA1 Channel 0 global interrupt
DMA1_Channel1_IRQn	DMA1 Channel 1 global interrupt
DMA1_Channel2_IRQn	DMA1 Channel 2 global interrupt
DMA1_Channel3_IRQn	DMA1 Channel 3 global interrupt
DMA1_Channel4_IRQn	DMA1 Channel 4 global interrupt
ENET_IRQn	ENET global interrupt
ENET_WKUP_IRQn	ENET wakeup through EXTI line interrupt
CAN1_TX_IRQn	CAN1 TX interrupt
CAN1_RX0_IRQn	CAN1 RX0 interrupt
CAN1_RX1_IRQn	CAN1 RX1 interrupt
CAN1_EWMC_IRQn	CAN1 EWMC interrupt
USBFS_IRQn	USBFS global interrupt
DMA1_Channel5_IRQn	DMA1 Channel 5 global interrupt
DMA1_Channel6_IRQn	DMA1 Channel 6 global interrupt
USART5_IRQn	USART5 global interrupt
I2C2_EV_IRQn	I2C2 event interrupt
I2C2_ER_IRQn	I2C2 error interrupt
DCI_IRQn	DCI global interrupt
CAU_IRQn	CAU global interrupt
HAU_TRNG_IRQn	HAU and TRNG global interrupt

UART6_IRQn	UART6 global interrupt
UART7_IRQn	UART7 global interrupt
TLI_IRQn	TLI global interrupt
TLI_ER_IRQn	TLI global error interrupt

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

Table 3-545. MISC firmware function

Function name	Function description
<code>nvic_priority_group_set</code>	set the priority group
<code>nvic_irq_enable</code>	enable NVIC interrupt request
<code>nvic_irq_disable</code>	disable NVIC interrupt request
<code>nvic_vector_table_set</code>	set the NVIC vector table base address
<code>system_lowpower_set</code>	set the state of the low power mode
<code>system_lowpower_reset</code>	reset the state of the low power mode
<code>systick_clksource_set</code>	set the systick clock source

nvic_priority_group_set

The description of `nvic_priority_group_set` is shown as below:

Table 3-546. Function `nvic_priority_group_set`

Function name	<code>nvic_priority_group_set</code>
Function prototype	<code>void nvic_priority_group_set(uint32_t nvic_prigroup);</code>
Function descriptions	set the priority group
Precondition	-
The called functions	-
Input parameter{in}	
nvic_prigroup	priority group
<i>NVIC_PRIGROUP_PR E0_SUB4</i>	0 bits for pre-emption priority 4 bits for subpriority
<i>NVIC_PRIGROUP_PR E1_SUB3</i>	1 bits for pre-emption priority 3 bits for subpriority
<i>NVIC_PRIGROUP_PR E2_SUB2</i>	2 bits for pre-emption priority 2 bits for subpriority
<i>NVIC_PRIGROUP_PR E3_SUB1</i>	3 bits for pre-emption priority 1 bits for subpriority
<i>NVIC_PRIGROUP_PR E4_SUB0</i>	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 nvic_irq_enable is shown as below:

Table 3-547. Function nvic_irq_enable

Function name	nvic_irq_enable
Function prototype	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
Function descriptions	enable NVIC interrupt request
Precondition	-
The called functions	nvic_priority_group_set
Input parameter{in}	
nvic_irq	NVIC interrupt, refer to enum Table 3-544. IRQn_Type
Input parameter{in}	
nvic_irq_pre_priority	the pre-emption priority needed to set (0~4)
Input parameter{in}	
nvic_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 */
```

```
nvic_irq_enable(WWDGT_IRQn, 1, 1);
```

nvic_irq_disable

The description of nvic_irq_disable is shown as below:

Table 3-548. Function nvic_irq_disable

Function name	nvic_irq_disable
Function prototype	void nvic_irq_disable(uint8_t nvic_irq);
Function descriptions	disable NVIC interrupt request
Precondition	-
The called functions	-
Input parameter{in}	
nvic_irq	NVIC interrupt, refer to enum Table 3-544. 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 3-549. Function nvic_vector_table_set

Function name	nvic_vector_table_set
Function prototype	void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);
Function descriptions	set the NVIC vector table base address
Precondition	-
The called functions	-
Input parameter{in}	
nvic_vect_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 */
```

```
#define NVIC_VECTTAB_FLASH ((uint32_t)0x08000000)
```

```
nvic_vector_table_set(NVIC_VECTTAB_FLASH, 0x200);
```

system_lowpower_set

The description of system_lowpower_set is shown as below:

Table 3-550. Function system_lowpower_set

Function name	system_lowpower_set
Function prototype	void system_lowpower_set(uint8_t lowpower_mode);
Function descriptions	set the state of the low power mode
Precondition	-
The called functions	-

Input parameter{in}	
lowpower_mode	the low power mode state
SCB_LPM_SLEEP_EXIT_ISR	if chose this para, the system always enter low power mode by exiting from ISR
SCB_LPM_DEEPSLEEP_P	if chose this para, the system will enter the DEEPSLEEP mode
SCB_LPM_WAKE_BY_ALL_INT	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 */

#define SCB_SCR_SLEEPONEXIT ((uint8_t)0x02)

#define SCB_LPM_SLEEP_EXIT_ISR SCB_SCR_SLEEPONEXIT

system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);

```

system_lowpower_reset

The description of system_lowpower_reset is shown as below:

Table 3-551. Function system_lowpower_reset

Function name	system_lowpower_reset
Function prototype	void system_lowpower_reset(uint8_t lowpower_mode);
Function descriptions	reset the state of the low power mode
Precondition	-
The called functions	-
Input parameter{in}	
lowpower_mode	the low power mode state
SCB_LPM_SLEEP_EXIT_ISR	if chose this para, the system will exit low power mode by exiting from ISR
SCB_LPM_DEEPSLEEP_P	if chose this para, the system will enter the SLEEP mode
SCB_LPM_WAKE_BY_ALL_INT	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 */
```

```
#define SCB_SCR_SLEEPONEXIT ((uint8_t)0x02)
```

```
#define SCB_LPM_SLEEP_EXIT_ISR SCB_SCR_SLEEPONEXIT
```

```
system_lowpower_reset(SCB_LPM_SLEEP_EXIT_ISR);
```

systick_clksource_set

The description of systick_clksource_set is shown as below:

Table 3-552. 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 */
```

```
#define SYSTICK_CLKSOURCE_HCLK_DIV8 ((uint32_t)0xFFFFFFFFBU)
```

```
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 3-553. 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 3-554. PMU firmware function

Function name	Function description
pmu_deinit	reset PMU registers
pmu_lvd_select	select low voltage detector threshold
pmu_lvd_disable	disable PMU lvd
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 PMU wakeup pin
pmu_wakeup_pin_disable	disable PMU wakeup pin
pmu_backup_write_enable	enable write access to the registers in backup domain
pmu_backup_write_disable	disable write access to the registers in backup domain
pmu_flag_get	get PMU flag
pmu_flag_clear	clear PMU flag

pmu_deinit

The description of pmu_deinit is shown as below:

Table 3-555. Function pmu_deinit

Function name	pmu_deinit
Function prototype	void pmu_deinit(void);
Function descriptions	reset PMU registers
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 3-556. Function pmu_lvd_select

Function name	pmu_lvd_select
Function prototype	void pmu_lvd_select(uint32_t lvd_tn);
Function descriptions	select low voltage detector threshold
Precondition	-
The called functions	-
Input parameter{in}	
lvd_tn	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 2.9V */
```

```
pmu_lvd_select (PMU_LVDT_7);
```

pmu_lvd_disable

The description of pmu_lvd_disable is shown as below:

Table 3-557. 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_to_sleepmode

The description of pmu_to_sleepmode is shown as below:

Table 3-558. 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
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PMU work at sleep mode */
pmu_to_sleepmode (WFI_CMD);
```

pmu_to_deepsleepmode

The description of pmu_to_deepsleepmode is shown as below:

Table 3-559. 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
<i>PMU_LDO_NORMAL</i>	LDO work at normal power mode when pmu enter deepsleep mode
<i>PMU_LDO_LOWPOWER</i>	LDO work at low power mode when pmu enter deepsleep mode

Input parameter{in}	
deepsleepmodecmd	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:

```
/* PMU work at deepsleep mode */
```

```
pmu_to_deepsleepmode (PMU_LDO_NORMAL, WFI_CMD);
```

pmu_to_standbymode

The description of pmu_to_standbymode is shown as below:

Table 3-560. 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:

```
/* PMU work at standby mode */
```

```
pmu_to_standby ();
```

pmu_wakeup_pin_enable

The description of pmu_wakeup_pin_enable is shown as below:

Table 3-561. Function pmu_wakeup_pin_enable

Function name	pmu_wakeup_pin_enable
Function prototype	void pmu_wakeup_pin_enable(void);
Function descriptions	enable PMU 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 3-562. Function pmu_wakeup_pin_disable

Function name	pmu_wakeup_pin_disable
Function prototype	void pmu_wakeup_pin_disable (void);
Function descriptions	disable PMU wakeup pin
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable wakeup pin */
pmu_wakeup_pin_disable ();
```

pmu_backup_write_enable

The description of pmu_backup_write_enable is shown as below:

Table 3-563. Function pmu_backup_write_enable

Function name	pmu_backup_write_enable
Function prototype	void pmu_backup_write_enable (void);
Function descriptions	enable write access to the registers in backup domain
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable write access to the registers in backup domain */
```

```
pmu_backup_write_enable ();
```

pmu_backup_write_disable

The description of pmu_backup_write_disable is shown as below:

Table 3-564. Function pmu_backup_write_disable

Function name	pmu_backup_write_disable
Function prototype	void pmu_backup_write_disable (void);
Function descriptions	disable write access to the registers in backup domain
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable write access to the registers in backup domain */
```

```
pmu_backup_write_disable ();
```

pmu_flag_get

The description of pmu_flag_get is shown as below:

Table 3-565. Function pmu_flag_get

Function name	pmu_flag_get
Function prototype	FlagStatus pmu_flag_get(uint32_t flag);
Function descriptions	get PMU flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	PMU flag
PMU_FLAG_WAKEUP	wakeup flag
PMU_FLAG_STANDBY	standby flag

<i>PMU_FLAG_LVD</i>	lvd flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get PMU wakeup flag */
```

```
FlagStatus status;
```

```
status = pmu_flag_get (PMU_FLAG_WAKEUP);
```

pmu_flag_clear

The description of pmu_flag_clear is shown as below:

Table 3-566. Function pmu_flag_clear

Function name	pmu_flag_clear
Function prototype	void pmu_flag_clear(uint32_t flag_reset);
Function descriptions	clear PMU flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	PMU flag
<i>PMU_FLAG_RESET_WAKEUP</i>	reset wakeup flag
<i>PMU_FLAG_RESET_S TANDBY</i>	reset standby flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear PMU wakeup flag */
```

```
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

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 3-567. RCU Registers

Registers	Descriptions
RCU_CTL	control register
RCU_CFG0	clock configuration register 0
RCU_INT	clock interrupt register
RCU_APB2RST	APB2 reset register
RCU_APB1RST	APB1 reset register
RCU_AHB1EN	AHB1 enable register
RCU_APB2EN	APB2 enable register
RCU_APB1EN	APB1 enable register
RCU_BDCTL	Backup domain control register
RCU_RSTSCK	reset source/clock register
RCU_AHB1RST	AHB1 reset register
RCU_CFG1	configuration register 1
RCU_DSV	Deep-sleep mode voltage register
RCU_AHB2EN	AHB2 enable register
RCU_ADDAPB2EN	APB2 additional enable register
RCU_ADDAPB1EN	APB1 additional enable register
RCU_AHB2RST	AHB2 reset register
RCU_ADDAPB2RST	APB2 additional reset register
RCU_ADDAPB1RST	APB1 additional reset register
RCU_CFG2	configuration register 2
RCU_PLLTCTL	PLLT control register
RCU_PLLTINT	PLLT interrupt register
RCU_PLLTCFG	PLLT configuration register

3.21.2. Descriptions of Peripheral functions

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

Table 3-568. 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 sleep mode
rcu_periph_clock_sleep_disable	disable the peripherals clock when sleep mode

Function name	Function description
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
rcu_pll_config	configure the main PLL clock
rcu_predv0_config	configure the PREDV0 division factor
rcu_predv1_config	configure the PREDV1 division factor
rcu_pll1_config	configure the PLL1 clock
rcu_pll2_config	configure the PLL2 clock
rcu_adc_clock_config	configure the ADC prescaler factor
rcu_usbfs_trng_clock_config	configure the USBFS/TRNG prescaler factor
rcu_rtc_clock_config	configure the RTC clock source selection
rcu_i2s1_clock_config	configure the I2S1 clock source selection
rcu_i2s2_clock_config	configure the I2S2 clock source selection
rcu_pllt_config	configure the PLLT clock selection
rcu_pllt_vco_config	Configure the PLLT clock multiplication and division factors
rcu_tli_clock_config	configure the TLI prescaler factor
rcu_lxtal_drive_capability_config	configure the LXTAL drive capability
rcu_osci_stab_wait	wait for oscillator stabilization flags is SET or oscillator startup is timeout
rcu_osci_on	turn on the oscillator
rcu_osci_off	turn off the oscillator
rcu_osci_bypass_mode_enable	enable the oscillator bypass mode
rcu_osci_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_irc8m_adjust_value_set	set the IRC8M adjust value
rcu_deepsleep_voltage_set	set the deep-sleep mode voltage value
rcu_clock_freq_get	get the system clock, bus clock frequency
rcu_flag_get	get the clock stabilization and peripheral reset flags
rcu_all_reset_flag_clear	clear all the reset flag
rcu_interrupt_enable	enable the stabilization interrupt
rcu_interrupt_disable	disable the stabilization interrupt
rcu_interrupt_flag_get	get the clock stabilization interrupt and ckm flags

Function name	Function description
rcu_interrupt_flag_clear	clear the interrupt flags

Enum rcu_periph_enum

Table 3-569. Enum rcu_periph_enum

Member name	Function description
RCU_GPIOx	GPIO ports clock (x=A,B,C,D,E,F,G ,H,I)
RCU_AF	alternate function clock
RCU_CRC	CRC clock
RCU_DMAx	DMAx clock (x=0,1)
RCU_ENET	ENET clock
RCU_ENETTX	ENETTX clock
RCU_ENETRX	ENETRX 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)
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
RCU_BKPI	BKP interface clock
RCU_TLI	TLI clock
RCU_DCI	DCI clock
RCU_CAU	CAU clock
RCU_HAU	HAU clock
RCU_TRNG	TRNG clock

Enum rcu_periph_sleep_enum

Table 3-570. Enum rcu_periph_sleep_enum

Member name	Function description
RCU_FMC_SLP	FMC clock
RCU_SRAM_SLP	SRAM clock

Enum rcu_periph_reset_enum

Table 3-571. Enum rcu_periph_reset_enum

Member name	Function description
RCU_GPIOxRST	reset GPIO ports clock (x=A,B,C,D,E,F,G,H,I)
RCU_AFRST	reset alternate function clock
RCU_ENETRST	reset ENET clock
RCU_USBFIRST	reset USBFS 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)
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_BKPIRST	reset BKPI clock
RCU_TLIRST	reset TLI
RCU_DCIRST	reset DCI
RCU_CAURST	reset CAU
RCU_HAURST	reset HAU
RCU_TRNGRST	reset TRNG

Enum rcu_flag_enum

Table 3-572. Enum rcu_flag_enum

Member name	Function description
RCU_FLAG_IRC8M STB	IRC8M stabilization flag
RCU_FLAG_HXTAL STB	HXTAL stabilization flag
RCU_FLAG_PLLST B	PLL stabilization flag
RCU_FLAG_PLL1S TB	PLL1 stabilization flag
RCU_FLAG_PLL2S TB	PLL2 stabilization flag
RCU_FLAG_PLLTS TB	PLLT stabilization flag
RCU_FLAG_LXTAL STB	LXTAL stabilization flag

Member name	Function description
RCU_FLAG_IRC40KSTB	IRC40K stabilization flag
RCU_FLAG_EPRST	external PIN reset flag
RCU_FLAG_PORRST	power reset flag
RCU_FLAG_SWRST	software reset flag
RCU_FLAG_FWDGTRST	free watchdog timer reset flag
RCU_FLAG_WWDGTRST	window watchdog timer reset flag
RCU_FLAG_LPRST	low-power reset flag

Enum rcu_int_flag_enum

Table 3-573. Enum rcu_int_flag_enum

Member name	Function description
RCU_INT_FLAG_IRC40KSTB	IRC40K stabilization interrupt flag
RCU_INT_FLAG_LXTALSTB	LXTAL stabilization interrupt flag
RCU_INT_FLAG_IRC8MSTB	IRC8M stabilization interrupt flag
RCU_INT_FLAG_HXTALSTB	HXTAL stabilization interrupt flag
RCU_INT_FLAG_PLLSTB	PLL stabilization interrupt flag
RCU_INT_FLAG_PLL1STB	PLL1 stabilization interrupt flag
RCU_INT_FLAG_PLL2STB	PLL2 stabilization interrupt flag
RCU_INT_FLAG_CKMK	HXTAL clock stuck interrupt flag
RCU_INT_FLAG_PLTSTB	PLLT stabilization interrupt flag

Enum rcu_int_flag_clear_enum

Table 3-574. Enum rcu_int_flag_clear_enum

Member name	Function description
RCU_INT_FLAG_IRC40KSTB_CLR	IRC40K stabilization interrupt flag clear

Member name	Function description
RCU_INT_FLAG_L XTALSTB_CLR	LXTAL stabilization interrupt flag clear
RCU_INT_FLAG_IR C8MSTB_CLR	IRC8M stabilization interrupt flag clear
RCU_INT_FLAG_H XTALSTB_CLR	HXTAL stabilization interrupt flag clear
RCU_INT_FLAG_P LLSTB_CLR	PLL stabilization interrupt flag clear
RCU_INT_FLAG_P LL1STB_CLR	PLL1 stabilization interrupt flag clear
RCU_INT_FLAG_P LL2STB_CLR	PLL2 stabilization interrupt flag clear
RCU_INT_FLAG_C KM_CLR	clock stuck interrupt flag clear
RCU_INT_FLAG_P LLTSTB_CLR	PLLT stabilization interrupt flag clear

Enum rcu_int_enum

Table 3-575. Enum rcu_int_enum

Member name	Function description
RCU_INT_IRC40KS TB	IRC40K stabilization interrupt enable
RCU_INT_LXTALS TB	LXTAL stabilization interrupt enable
RCU_INT_IRC8MS TB	IRC8M stabilization interrupt enable
RCU_INT_HXTALS TB	HXTAL stabilization interrupt enable
RCU_INT_PLLSTB	PLL stabilization interrupt enable
RCU_INT_PLL1STB	PLL1 stabilization interrupt enable
RCU_INT_PLL2STB	PLL2 stabilization interrupt enable
RCU_INT_PLLTST B	PLLT stabilization interrupt enable

Enum rcu_osci_type_enum

Table 3-576. Enum rcu_osci_type_enum

Member name	Function description
RCU_IRC8M	internal 8M RC oscillators(IRC8M)
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_PLL_CK	phase locked loop(PLL)
RCU_PLL1_CK	phase locked loop 1

Member name	Function description
RCU_PLL2_CK	phase locked loop 2
RCU_LXTAL	low speed crystal oscillator(LXTAL)
RCU_IRC40K	internal 40K RC oscillator(IRC40K)
RCU_PLLT_CK	TLI phase locked loop

Enum rcu_clock_freq_enum

Table 3-577. Enum rcu_clock_freq_enum

Member 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 3-578. Function rcu_deinit

Function name	rcu_deinit
Function prototype	void rcu_deinit(void);
Function descriptions	deinitialize the RCU
Precondition	-
The called functions	rcu_osci_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 3-579. 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 3-569. Enum rcu_periph enum.
<i>RCU_GPIOx</i>	GPIO ports clock (x=A,B,C,D,E,F,G ,H,I)
<i>RCU_AF</i>	alternate function clock
<i>RCU_CRC</i>	CRC clock
<i>RCU_DMAx</i>	DMAx clock (x=0,1)
<i>RCU_ENET</i>	ENET clock
<i>RCU_ENETTX</i>	ENETTX clock
<i>RCU_ENETRX</i>	ENETRX clock
<i>RCU_USBFS</i>	USBFS clock
<i>RCU_EXMC</i>	EXMC clock
<i>RCU_TIMERx</i>	TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGT</i>	WWDGT clock
<i>RCU_SPIx</i>	SPIx clock (x=0,1,2)
<i>RCU_USARTx</i>	USARTx clock (x=0,1,2,5)
<i>RCU_UARTx</i>	UARTx clock (x=3,4 ,6,7)
<i>RCU_I2Cx</i>	I2Cx clock (x=0,1,2)
<i>RCU_CANx</i>	CANx clock (x=0,1)
<i>RCU_PMU</i>	PMU clock
<i>RCU_DAC</i>	DAC clock
<i>RCU_RTC</i>	RTC clock
<i>RCU_ADCx</i>	ADCx clock (x=0,1,2)
<i>RCU_SDIO</i>	SDIO clock
<i>RCU_BKPI</i>	BKP interface clock
<i>RCU_TLI</i>	TLI clock
<i>RCU_DCI</i>	DCI clock
<i>RCU_CAU</i>	CAU clock
<i>RCU_HAU</i>	HAU clock
<i>RCU_TRNG</i>	TRNG clock
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 3-580. 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 3-569. Enum rcu_periph_enum.
<i>RCU_GPIOx</i>	GPIO ports clock (x=A,B,C,D,E,F,G ,H,I).
<i>RCU_AF</i>	alternate function clock
<i>RCU_CRC</i>	CRC clock
<i>RCU_DMAx</i>	DMAx clock (x=0,1)
<i>RCU_ENET</i>	ENET clock
<i>RCU_ENETTX</i>	ENETTX clock
<i>RCU_ENETRX</i>	ENETRX clock
<i>RCU_USBFS</i>	USBFS clock
<i>RCU_EXMC</i>	EXMC clock
<i>RCU_TIMERx</i>	TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGT</i>	WWDGT clock
<i>RCU_SPIx</i>	SPIx clock (x=0,1,2)
<i>RCU_USARTx</i>	USARTx clock (x=0,1,2,5)
<i>RCU_UARTx</i>	UARTx clock (x=3,4 ,6,7)
<i>RCU_I2Cx</i>	I2Cx clock (x=0,1,2)
<i>RCU_CANx</i>	CANx clock (x=0,1)
<i>RCU_PMU</i>	PMU clock
<i>RCU_DAC</i>	DAC clock
<i>RCU_RTC</i>	RTC clock
<i>RCU_ADCx</i>	ADCx clock (x=0,1,2)
<i>RCU_SDIO</i>	SDIO clock
<i>RCU_BKPI</i>	BKP interface clock
<i>RCU_TLI</i>	TLI clock
<i>RCU_DCI</i>	DCI clock
<i>RCU_CAU</i>	CAU clock
<i>RCU_HAU</i>	HAU clock
<i>RCU_TRNG</i>	TRNG clock
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 3-581. 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 3-570. Enum rcu_periph_sleep_enum .
<i>RCU_FMC_SLP</i>	FMC clock
<i>RCU_SRAM_SLP</i>	SRAM clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the FMC clock when in sleep mode */
```

```
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

rcu_periph_clock_sleep_disable

The description of rcu_periph_clock_sleep_disable is shown as below:

Table 3-582. 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 3-570. Enum rcu_periph_sleep_enum .
<i>RCU_FMC_SLP</i>	FMC clock
<i>RCU_SRAM_SLP</i>	SRAM clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the FMC clock when in sleep mode */
```

```
rcu_periph_clock_sleep_disable(RCU_FMC_SLP);
```

rcu_periph_reset_enable

The description of rcu_periph_reset_enable is shown as below:

Table 3-583. 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 3-571. Enum rcu_periph_reset_enum .
<i>RCU_GPIOxRST</i>	reset GPIO ports clock (x=A,B,C,D,E,F,G,H,I)
<i>RCU_AFRST</i>	reset alternate function clock
<i>RCU_ENETRST</i>	reset ENET clock
<i>RCU_USBFSRST</i>	reset USBFS clock
<i>RCU_TIMERxRST</i>	reset TIMEx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGTRST</i>	reset WWDGT clock
<i>RCU_SPIxRST</i>	reset SPIx clock (x=0,1,2)
<i>RCU_USARTxRST</i>	reset USARTx clock (x=0,1,2,5)
<i>RCU_UARTxRST</i>	reset UARTx clock (x=3,4,6,7)
<i>RCU_I2CxRST</i>	reset I2Cx clock (x=0,1,2)
<i>RCU_CANxRST</i>	reset CANx clock (x=0,1)
<i>RCU_PMURST</i>	reset PMU clock
<i>RCU_DACRST</i>	reset DAC clock
<i>RCU_ADCxRST</i>	reset ADCx clock (x=0,1,2)
<i>RCU_BKPIRST</i>	reset BKPI clock
<i>RCU_TLIRST</i>	reset TLI
<i>RCU_DCIRST</i>	reset DCI
<i>RCU_CAURST</i>	reset CAU
<i>RCU_HAURST</i>	reset HAU
<i>RCU_TRNGRST</i>	reset TRNG
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 3-584. 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 3-571. Enum rcu_periph_reset_enum .
<i>RCU_GPIOxRST</i>	reset GPIO ports clock (x=A,B,C,D,E,F,G,H,I)
<i>RCU_AFRST</i>	reset alternate function clock
<i>RCU_ENETRST</i>	reset ENET clock
<i>RCU_USBFIRST</i>	reset USBFS clock
<i>RCU_TIMERxRST</i>	reset TIMEx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGTRST</i>	reset WWDGT clock
<i>RCU_SPIxRST</i>	reset SPIx clock (x=0,1,2)
<i>RCU_USARTxRST</i>	reset USARTx clock (x=0,1,2,5)
<i>RCU_UARTxRST</i>	reset UARTx clock (x=3,4,6,7)
<i>RCU_I2CxRST</i>	reset I2Cx clock (x=0,1,2)
<i>RCU_CANxRST</i>	reset CANx clock (x=0,1)
<i>RCU_PMURST</i>	reset PMU clock
<i>RCU_DACRST</i>	reset DAC clock
<i>RCU_ADCxRST</i>	reset ADCx clock (x=0,1,2)
<i>RCU_BKPIRST</i>	reset BKPI clock
<i>RCU_TLIRST</i>	reset TLI
<i>RCU_DCIRST</i>	reset DCI
<i>RCU_CAURST</i>	reset CAU
<i>RCU_HAURST</i>	reset HAU
<i>RCU_TRNGRST</i>	reset TRNG
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 3-585. 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 3-586. 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 3-587. 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_CKSYSSRC_IRC8M</i>	select CK_IRC8M as the CK_SYS source
<i>RCU_CKSYSSRC_HXTAL</i>	select CK_HXTAL as the CK_SYS source
<i>RCU_CKSYSSRC_PLL</i>	select CK_PLL 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_CKSYSSRC_HXTAL);
```

rcu_system_clock_source_get

The description of rcu_system_clock_source_get is shown as below:

Table 3-588. 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	
uint32_t	which clock is selected as CK_SYS source
<i>RCU_SCSS_IRC8M</i>	CK_IRC8M is selected as the CK_SYS source
<i>RCU_SCSS_HXTAL</i>	CK_HXTAL is selected as the CK_SYS source

<i>RCU_SCSS_PLL</i>	CK_PLL 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 3-589. Function rcu_ahb_clock_config

Function name	rcu_ahb_clock_config
Function prototype	void rcu_ahb_clock_config(uint32_t ck_ahb);
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_DIVx</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 3-590. 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>	select CK_AHB as CK_APB1

<i>IV1</i>	
<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 3-591. 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_D</i> <i>IV1</i>	select CK_AHB as CK_APB2
<i>RCU_APB2_CKAHB_D</i> <i>IV2</i>	select CK_AHB/2 as CK_APB2
<i>RCU_APB2_CKAHB_D</i> <i>IV4</i>	select CK_AHB/4 as CK_APB2
<i>RCU_APB2_CKAHB_D</i> <i>IV8</i>	select CK_AHB/8 as CK_APB2
<i>RCU_APB2_CKAHB_D</i> <i>IV16</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_CK_AHB_DIV8);
```

rcu_ckout0_config

The description of rcu_ckout0_config is shown as below:

Table 3-592. 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
RCU_CKOUT0SRC_NONE	no clock selected
RCU_CKOUT0SRC_CKSYS	select system clock CK_SYS
RCU_CKOUT0SRC_IRC8M	select high speed 8M internal oscillator clock
RCU_CKOUT0SRC_HXTAL	select HXTAL
RCU_CKOUT0SRC_CKPLL_DIV2	select (CK_PLL / 2) clock
RCU_CKOUT0SRC_CKPLL1	select CK_PLL1 clock
RCU_CKOUT0SRC_CKPLL2_DIV2	select (CK_PLL2 / 2) clock
RCU_CKOUT0SRC_EXT1	select EXT1 clock
RCU_CKOUT0SRC_CKPLL2	select CK_PLL2 clock
Input parameter{in}	
ckout0_div	CK_OUT0 divider
RCU_CKOUT0_DIVx(x = 1..64)	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 3-593. 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
RCU_CKOUT1SRC_NONE	no clock selected
RCU_CKOUT1SRC_CKSYS	select system clock CK_SYS
RCU_CKOUT1SRC_IRC8M	select high speed 8M internal oscillator clock
RCU_CKOUT1SRC_HXTAL	select HXTAL
RCU_CKOUT1SRC_CKPLL_DIV2	select (CK_PLL / 2) clock
RCU_CKOUT0SRC_CKPLL1	select CK_PLL1 clock
RCU_CKOUT1SRC_CKPLL2	select (CK_PLL2 / 2) clock
RCU_CKOUT1SRC_EXT1	select EXT1 clock
RCU_CKOUT1SRC_CKPLL2	select CK_PLL2 clock
Input parameter{in}	
ckout1_div	CK_OUT1 divider
RCU_CKOUT1_DIVx(x = 1..64)	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 3-594. Function rcu_pll_config

Function name	rcu_pll_config
Function prototype	void rcu_pll_config(uint32_t pll_src, uint32_t pll_mul);
Function descriptions	configure the main PLL clock
Precondition	-
The called functions	-
Input parameter{in}	
pll_src	PLL clock source selection
<i>RCU_PLLSRC_IRC8M_DIV2</i>	IRC8M/2 clock is selected as source clock of PLL
<i>RCU_PLLSRC_HXTAL</i>	HXTAL is selected as source clock of PLL
Input parameter{in}	
pll_mul	PLL clock multiplication factor
<i>RCU_PLL_MULx</i>	x = 2..14,16..32,6.5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLL */
rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PLL_MUL10);
```

rcu_predv0_config

The description of rcu_predv0_config is shown as below:

Table 3-595. Function rcu_predv0_config

Function name	rcu_predv0_config
Function prototype	void rcu_predv0_config(uint32_t predv0_source, uint32_t predv0_div);
Function descriptions	configure the PREDV0 division factor
Precondition	-
The called functions	-
Input parameter{in}	

predv0_source	PREDV0 input clock source selection
<i>RCU_PREDV0SRC_HXTAL</i>	select HXTAL as PREDV0 input source clock
<i>RCU_PREDV0SRC_CKPLL1</i>	select CK_PLL1 as PREDV0 input source clock
Input parameter{in}	
predv0_div	PREDV0 division factor
<i>RCU_PREDV0_DIVx</i>	PREDV0 input source clock is divided x (x=1..16)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PREDV0 division factor */
```

```
rcu_predv0_config(RCU_PREDV0SRC_HXTAL, RCU_PREDV0_DIV4);
```

rcu_predv1_config

The description of rcu_predv1_config is shown as below:

Table 3-596. Function rcu_predv1_config

Function name	rcu_predv1_config
Function prototype	void rcu_predv1_config(uint32_t predv1_div);
Function descriptions	configure the PREDV1 division factor
Precondition	-
The called functions	-
Input parameter{in}	
predv1_div	PREDV1 division factor
<i>RCU_PREDV1_DIVx</i>	PREDV1 input source clock is divided x (x=1..16)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PREDV1 division factor */
```

```
rcu_predv1_config(RCU_PREDV1_DIV8);
```

rcu_pll1_config

The description of rcu_pll1_config is shown as below:

Table 3-597. Function rcu_pll1_config

Function name	rcu_pll1_config
Function prototype	void rcu_pll1_config(uint32_t pll_mul);
Function descriptions	configure the PLL1 clock
Precondition	-
The called functions	-
Input parameter{in}	
pll_mul	PLL clock multiplication factor
RCU_PLL1_MULx	PLL1 clock * x, (x = 8..16, 20)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLL1 clock */
rcu_pll1_config(RCU_PLL1_MUL8);
```

rcu_pll2_config

The description of rcu_pll2_config is shown as below:

Table 3-598. Function rcu_pll2_config

Function name	rcu_pll2_config
Function prototype	void rcu_pll2_config(uint32_t pll_mul)
Function descriptions	configure the PLL2 clock
Precondition	-
The called functions	-
Input parameter{in}	
pll_mul	PLL clock multiplication factor
RCU_PLL2_MULx	PLL2 clock * x, (x = 8..16, 20)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLL2 clock */
rcu_pll2_config(RCU_PLL2_MUL8);
```

rcu_adc_clock_config

The description of rcu_adc_clock_config is shown as below:

Table 3-599. Function rcu_adc_clock_config

Function name	rcu_adc_clock_config
Function prototype	void rcu_adc_clock_config(uint32_t adc_psc);
Function descriptions	configure the ADC prescaler factor
Precondition	-
The called functions	-
Input parameter{in}	
adc_psc	ADC prescaler factor
<i>RCU_CKADC_CKAPB2_DIV2</i>	$CK_ADC = CK_APB2 / 2$
<i>RCU_CKADC_CKAPB2_DIV4</i>	$CK_ADC = CK_APB2 / 4$
<i>RCU_CKADC_CKAPB2_DIV6</i>	$CK_ADC = CK_APB2 / 6$
<i>RCU_CKADC_CKAPB2_DIV8</i>	$CK_ADC = CK_APB2 / 8$
<i>RCU_CKADC_CKAPB2_DIV12</i>	$CK_ADC = CK_APB2 / 12$
<i>RCU_CKADC_CKAPB2_DIV16</i>	$CK_ADC = CK_APB2 / 16$
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the ADC prescaler factor */
rcu_adc_clock_config(RCU_CKADC_CKAPB2_DIV8);
```

rcu_usbfs_trng_clock_config

The description of rcu_usbfs_trng_clock_config is shown as below:

Table 3-600. Function rcu_usbfs_trng_clock_config

Function name	rcu_usbfs_trng_clock_config
Function prototype	void rcu_usbfs_trng_clock_config(uint32_t usbfs_trng_psc);
Function descriptions	configure the USBFS/TRNG prescaler factor
Precondition	-
The called functions	-
Input parameter{in}	
usbfs_trng_psc	USBFS/TRNG prescaler factor
<i>RCU_CKUSB_CKPLL_DIV1_5</i>	USBFS/TRNG prescaler select CK_PLL/1.5

<i>RCU_CKUSB_CKPLL_DIV1</i>	USBFS/TRNG prescaler select CK_PLL/1
<i>RCU_CKUSB_CKPLL_DIV2_5</i>	USBFS/TRNG prescaler select CK_PLL/2.5
<i>RCU_CKUSB_CKPLL_DIV2</i>	USBFS/TRNG prescaler select CK_PLL/2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the USB prescaler factor */
```

```
rcu_usbfs_trng_clock_config(RCU_CKUSB_CKPLL_DIV2_5);
```

rcu_rtc_clock_config

The description of rcu_rtc_clock_config is shown as below:

Table 3-601. 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_IRC40K</i>	select CK_IRC40K as RTC source clock
<i>RCU_RTCSRC_HXTAL_DIV_128</i>	select CK_HXTAL/128 as RTC source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the RTC clock source selection */
```

```
rcu_rtc_clock_config(RCU_RTCSRC_IRC40K);
```

rcu_i2s1_clock_config

The description of rcu_i2s1_clock_config is shown as below:

Table 3-602. Function rcu_i2s1_clock_config

Function name	rcu_i2s1_clock_config
Function prototype	void rcu_i2s1_clock_config(uint32_t i2s_clock_source);
Function descriptions	configure the I2S1 clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
i2s_clock_source	I2S1 clock source selection
<i>RCU_I2S1SRC_CKSYS</i>	select system clock as I2S1 source clock
<i>RCU_I2S1SRC_CKPLL2_MUL2</i>	select CK_PLL2 * 2 as I2S1 source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the I2S1 clock source selection */
```

```
rcu_i2s1_clock_config(RCU_I2S1SRC_CKPLL2_MUL2);
```

rcu_i2s2_clock_config

The description of rcu_i2s2_clock_config is shown as below:

Table 3-603. Function rcu_i2s2_clock_config

Function name	rcu_i2s2_clock_config
Function prototype	void rcu_i2s2_clock_config(uint32_t i2s_clock_source);
Function descriptions	configure the I2S2 clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
i2s_clock_source	I2S2 clock source selection
<i>RCU_I2S2SRC_CKSYS</i>	select system clock as I2S2 source clock
<i>RCU_I2S2SRC_CKPLL2_MUL2</i>	select CK_PLL2 * 2 as I2S2 source clock
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure the I2S2 clock source selection */
```

```
rcu_i2s2_clock_config(RCU_I2S2SRC_CKPLL2_MUL2);
```

rcu_pllt_config

The description of rcu_pllt_config is shown as below:

Table 3-604. Function rcu_pllt_config

Function name	rcu_pllt_config
Function prototype	void rcu_pllt_config(uint32_t pllt_src);
Function descriptions	configure the PLLT clock selection
Precondition	-
The called functions	-
Input parameter{in}	
pllt_src	PLLT clock source selection
<i>RCU_PLLTSRC_IRC8M</i>	IRC8M selected as source clock of PLLT
<i>RCU_PLLTSRC_HXTAL</i>	HXTAL selected as source clock of PLLT
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLLT clock selection */
```

```
rcu_pllt_config(RCU_PLLTSRC_IRC8M);
```

rcu_pllt_vco_config

The description of rcu_pllt_vco_config is shown as below:

Table 3-605. Function rcu_pllt_vco_config

Function name	rcu_pllt_vco_config
Function prototype	ErrStatus rcu_pllt_vco_config(uint32_t pllt_psc, uint32_t pllt_mul, uint32_t plltr_psc);
Function descriptions	configure the PLLT clock multiplication and division factors
Precondition	-
The called functions	-
Input parameter{in}	
pllt_psc	the PLLT VCO input clock division factor

<i>parameter</i>	between 2 and 63
Input parameter{in}	
pllt_mul	the PLLT VCO output clock multiplication factor
<i>parameter</i>	between 49 and 432
Input parameter{in}	
ppltr_psc	the PLLTR division factor
<i>parameter</i>	between 2 and 7
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* configure the PLLT clock multiplication and division factors */
if(SUCCESS == rcu_pllt_vco_config(0x4, 0x64, 0x4)){
}

```

rcu_tli_clock_config

The description of rcu_tli_clock_config is shown as below:

Table 3-606. Function rcu_tli_clock_config

Function name	rcu_tli_clock_config
Function prototype	void rcu_tli_clock_config(uint32_t tli_psc);
Function descriptions	configure the TLI prescaler factor from PLLTR clock
Precondition	-
The called functions	-
Input parameter{in}	
tli_psc	TLI prescaler factor
<i>RCU_CKTLI_CKPLLTR_DIV2</i>	TLI prescaler select CK_PLLTR/2
<i>RCU_CKTLI_CKPLLTR_DIV4</i>	TLI prescaler select CK_PLLTR/4
<i>RCU_CKTLI_CKPLLTR_DIV8</i>	TLI prescaler select CK_PLLTR/8
<i>RCU_CKTLI_CKPLLTR_DIV16</i>	TLI prescaler select CK_PLLTR/16
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the TLI prescaler factor from PLLTR clock */
```

```
rcu_tli_clock_config(RCU_CKTLI_CKPLLTR_DIV2);
```

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	rcu_lxtal_drive_capability_config
Function prototype	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
Function descriptions	configure the LXTAL drive capability
Precondition	-
The called functions	-
Input parameter{in}	
lxtal_dricap	drive capability of LXTAL
<i>RCU_LXTAL_LOWDRI</i>	lower driving capability
<i>RCU_LXTAL_MED_LOWDRI</i>	medium low driving capability
<i>RCU_LXTAL_MED_HIGHDRI</i>	medium high driving capability
<i>RCU_LXTAL_HIGHDRI</i>	higher driving capability
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the LXTAL drive capability */
```

```
rcu_lxtal_drive_capability_config(RCU_LXTAL_LOWDRI);
```

rcu_osci_stab_wait

The description of rcu_osci_stab_wait is shown as below:

Table 3-608. Function rcu_osci_stab_wait

Function name	rcu_osci_stab_wait
Function prototype	ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci);
Function descriptions	wait for oscillator stabilization flags is SET or oscillator startup is timeout
Precondition	-
The called functions	rcu_flag_get
Input parameter{in}	
osci	oscillator types, refer to Table 3-576. Enum rcu_osci_type_enum .
<i>RCU_IRC8M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)

<i>RCU_PLL_CK</i>	phase locked loop(PLL)
<i>RCU_PLL1_CK</i>	phase locked loop 1
<i>RCU_PLL2_CK</i>	phase locked loop 2
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLLT_CK</i>	TLI phase locked loop
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* wait for oscillator stabilization flag */
if(SUCCESS == rcu_osci_stab_wait(RCU_HXTAL)){
}

```

rcu_osci_on

The description of rcu_osci_on is shown as below:

Table 3-609. Function rcu_osci_on

Function name	rcu_osci_on
Function prototype	void rcu_osci_on(rcu_osci_type_enum osci);
Function descriptions	turn on the oscillator
Precondition	-
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to Table 3-576. Enum rcu_osci_type_enum.
<i>RCU_IRC8M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_PLL_CK</i>	phase locked loop(PLL)
<i>RCU_PLL1_CK</i>	phase locked loop 1
<i>RCU_PLL2_CK</i>	phase locked loop 2
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLLT_CK</i>	TLI phase locked loop
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn on the high speed crystal oscillator */

```

```
rcu_osci_on(RCU_HXTAL);
```

rcu_osci_off

The description of rcu_osci_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 3-576. Enum rcu_osci_type_enum .
<i>RCU_IRC8M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_PLL_CK</i>	phase locked loop(PLL)
<i>RCU_PLL1_CK</i>	phase locked loop 1
<i>RCU_PLL2_CK</i>	phase locked loop 2
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLLT_CK</i>	TLI phase locked loop
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 3-576. Enum rcu_osci_type_enum .
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)

<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

rcu_osci_bypass_mode_disable

The description of rcu_osci_bypass_mode_disable is shown as below:

Table 3-612. Function rcu_osci_bypass_mode_disable

Function name	rcu_osci_bypass_mode_disable
Function prototype	void rcu_osci_bypass_mode_disable(rcu_osci_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 3-576. Enum rcu_osci_type_enum .
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_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_irc8m_adjust_value_set

The description of rcu_irc8m_adjust_value_set is shown as below:

Table 3-615. Function rcu_irc8m_adjust_value_set

Function name	rcu_irc8m_adjust_value_set
Function prototype	void rcu_irc8m_adjust_value_set(uint8_t irc8m_adjval);
Function descriptions	set the IRC8M adjust value
Precondition	-
The called functions	-
Input parameter{in}	
irc8m_adjval	IRC8M adjust value, must be between 0 and 0x1F
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* set the IRC8M adjust value */
rcu_irc8m_adjust_value_set(0x10);
```

rcu_deepsleep_voltage_set

The description of rcu_deepsleep_voltage_set is shown as below:

Table 3-616. Function rcu_deepsleep_voltage_set

Function name	rcu_deepsleep_voltage_set
Function prototype	void rcu_deepsleep_voltage_set(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_1_2	the core voltage is 1.2V in deep-sleep mode
RCU_DEEPSLEEP_V_1_1	the core voltage is 1.1V in deep-sleep mode
RCU_DEEPSLEEP_V_1_0	the core voltage is 1.0V in deep-sleep mode
RCU_DEEPSLEEP_V_0_9	the core voltage is 0.9V in deep-sleep mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the deep-sleep mode voltage */
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1_1);
```

rcu_clock_freq_get

The description of rcu_clock_freq_get is shown as below:

Table 3-617. 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}	
clock	the clock frequency which to get, refer to Table 3-577. Enum rcu_clock_freq enum.
<i>CK_SYS</i>	system clock frequency
<i>CK_AHB</i>	AHB clock frequency
<i>CK_APB1</i>	APB1 clock frequency
<i>CK_APB2</i>	APB2 clock frequency
Output parameter{out}	
-	-
Return value	
ck_freq	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);
```

rcu_flag_get

The description of rcu_flag_get is shown as below:

Table 3-618. 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 peripheral reset flags
Precondition	-
The called functions	-
Input parameter{in}	
flag	the clock stabilization and peripheral reset flags, refer to Table 3-572. Enum rcu_flag enum.
<i>RCU_FLAG_IRC8MSTB</i>	IRC8M stabilization flag
<i>RCU_FLAG_HXTALSTB</i>	HXTAL stabilization flag
<i>RCU_FLAG_PLLSTB</i>	PLL stabilization flag
<i>RCU_FLAG_PLL1STB</i>	PLL1 stabilization flag
<i>RCU_FLAG_PLL2STB</i>	PLL2 stabilization flag
<i>RCU_FLAG_PLTSTB</i>	PLLT stabilization flag
<i>RCU_FLAG_LXTALSTB</i>	LXTAL stabilization flag

<i>RCU_FLAG_IRC40KSTB</i>	IRC40K stabilization flag
<i>RCU_FLAG_EPRST</i>	external PIN reset flag
<i>RCU_FLAG_PORRST</i>	power reset flag
<i>RCU_FLAG_SWRST</i>	software reset flag
<i>RCU_FLAG_FWDGTRST</i>	free watchdog timer reset flag
<i>RCU_FLAG_WWDGTRST</i>	window watchdog timer reset flag
<i>RCU_FLAG_LPRST</i>	low-power reset flag
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 3-619. 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_enable

The description of rcu_interrupt_enable is shown as below:

Table 3-620. 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 3-575. Enum rcu_int_enum .
RCU_INT_IRC40KSTB	IRC40K stabilization interrupt enable
RCU_INT_LXTALSTB	LXTAL stabilization interrupt enable
RCU_INT_IRC8MSTB	IRC8M stabilization interrupt enable
RCU_INT_HXTALSTB	HXTAL stabilization interrupt enable
RCU_INT_PLLSTB	PLL stabilization interrupt enable
RCU_INT_PLL1STB	PLL1 stabilization interrupt enable
RCU_INT_PLL2STB	PLL2 stabilization interrupt enable
RCU_INT_PLTSTB	PLLT stabilization interrupt enable
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 3-621. 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 3-575. Enum rcu_int_enum .
RCU_INT_IRC40KSTB	IRC40K stabilization interrupt enable
RCU_INT_LXTALSTB	LXTAL stabilization interrupt enable
RCU_INT_IRC8MSTB	IRC8M stabilization interrupt enable

<i>RCU_INT_HXTALSTB</i>	HXTAL stabilization interrupt enable
<i>RCU_INT_PLLSTB</i>	PLL stabilization interrupt enable
<i>RCU_INT_PLL1STB</i>	PLL1 stabilization interrupt enable
<i>RCU_INT_PLL2STB</i>	PLL2 stabilization interrupt enable
<i>RCU_INT_PLLTSTB</i>	PLLT stabilization interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the HXTAL stabilization interrupt */
```

```
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

rcu_interrupt_flag_get

The description of `rcu_interrupt_flag_get` is shown as below:

Table 3-622. Function `rcu_interrupt_flag_get`

Function name	<code>rcu_interrupt_flag_get</code>
Function prototype	<code>FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);</code>
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 3-573. Enum <code>rcu_int_flag_enum</code> .
<i>RCU_INT_FLAG_IRC40KSTB</i>	IRC40K stabilization interrupt flag
<i>RCU_INT_FLAG_LXTALSTB</i>	LXTAL stabilization interrupt flag
<i>RCU_INT_FLAG_IRC8MSTB</i>	IRC8M stabilization interrupt flag
<i>RCU_INT_FLAG_HXTALSTB</i>	HXTAL stabilization interrupt flag
<i>RCU_INT_FLAG_PLLSTB</i>	PLL stabilization interrupt flag
<i>RCU_INT_FLAG_PLL1STB</i>	PLL1 stabilization interrupt flag
<i>RCU_INT_FLAG_PLL2STB</i>	PLL2 stabilization interrupt flag
<i>RCU_INT_FLAG_CKM</i>	HXTAL clock stuck interrupt flag
<i>RCU_INT_FLAG_PLLTSTB</i>	PLLT stabilization interrupt flag

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 3-623. 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 3-574. Enum rcu_int_flag_clear_enum .
RCU_INT_FLAG_IRC40KSTB_CLR	IRC40K stabilization interrupt flag clear
RCU_INT_FLAG_LXTALSTB_CLR	LXTAL stabilization interrupt flag clear
RCU_INT_FLAG_IRC8MSTB_CLR	IRC8M stabilization interrupt flag clear
RCU_INT_FLAG_HXTALSTB_CLR	HXTAL stabilization interrupt flag clear
RCU_INT_FLAG_PLLSTB_CLR	PLL stabilization interrupt flag clear
RCU_INT_FLAG_PLL1STB_CLR	PLL1 stabilization interrupt flag clear
RCU_INT_FLAG_PLL2STB_CLR	PLL2 stabilization interrupt flag clear
RCU_INT_FLAG_CKM_CLR	clock stuck interrupt flag clear
RCU_INT_FLAG_PLLOTSTB_CLR	PLLT stabilization interrupt flag clear
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* clear the interrupt HXTAL stabilization interrupt flag */
```

```
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

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 3-624. RTC Registers

Registers	Descriptions
RTC_INTEN	interrupt enable register
RTC_CTL	control register
RTC_PSCH	prescaler high register
RTC_PSCL	prescaler low register
RTC_DIVH	divider high register
RTC_DIVL	divider low register
RTC_CNTH	counter high register
RTC_CNTL	counter low register
RTC_ALRMH	alarm high register
RTC_ALRML	alarm low register

3.22.2. Descriptions of Peripheral functions

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

Table 3-625. RTC firmware function

Function name	Function description
rtc_configuration_mode_enter	enter RTC configuration mode
rtc_configuration_mode_exit	exit RTC configuration mode
rtc_lwoff_wait	wait RTC last write operation finished flag set
rtc_register_sync_wait	wait RTC registers synchronized flag set
rtc_counter_get	get RTC counter value
rtc_counter_set	set RTC counter value

Function name	Function description
rtc_prescaler_set	set RTC prescaler value
rtc_alarm_config	set RTC alarm value
rtc_divider_get	get RTC divider value
rtc_flag_get	get RTC flag status
rtc_flag_clear	clear RTC flag status
rtc_interrupt_enable	enable RTC interrupt
rtc_interrupt_disable	disable RTC interrupt

rtc_configuration_mode_enter

The description of rtc_configuration_mode_enter is shown as below:

Table 3-626. Function rtc_configuration_mode_enter

Function name	rtc_configuration_mode_enter
Function prototype	void rtc_configuration_mode_enter(void);
Function descriptions	enter RTC configuration mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enter RTC configuration mode */
rtc_configuration_mode_enter();
```

rtc_configuration_mode_exit

The description of rtc_configuration_mode_exit is shown as below:

Table 3-627. Function rtc_configuration_mode_exit

Function name	rtc_configuration_mode_exit
Function prototype	void rtc_configuration_mode_exit(void);
Function descriptions	exit RTC configuration mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* exit RTC configuration mode */
```

```
rtc_configuration_mode_exit();
```

rtc_lwoff_wait

The description of rtc_lwoff_wait is shown as below:

Table 3-628. Function rtc_lwoff_wait

Function name	rtc_lwoff_wait
Function prototype	void rtc_lwoff_wait(void);
Function descriptions	wait RTC last write operation finished flag set
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait();
```

rtc_register_sync_wait

The description of rtc_register_sync_wait is shown as below:

Table 3-629. Function rtc_register_sync_wait

Function name	rtc_register_sync_wait
Function prototype	void rtc_register_sync_wait(void);
Function descriptions	wait RTC registers synchronized flag set
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* wait for RTC registers synchronization */

rtc_register_sync_wait();
```

rtc_counter_get

The description of rtc_counter_get is shown as below:

Table 3-630. Function rtc_counter_get

Function name	rtc_counter_get
Function prototype	uint32_t rtc_counter_get(void);
Function descriptions	get RTC counter value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the value of RTC counter

Example:

```
/* get the counter value */

uint32_t rtc_counter_value;

rtc_counter_value = rtc_counter_get();
```

rtc_counter_set

The description of rtc_counter_set is shown as below:

Table 3-631. Function rtc_counter_set

Function name	rtc_counter_set
Function prototype	void rtc_counter_set(uint32_t cnt);
Function descriptions	set RTC counter value
Precondition	before using this function, you must call rtc_lwoff_wait() function (wait until LWOFF flag is set).
The called functions	rtc_configuration_mode_enter / rtc_configuration_mode_exit
Input parameter{in}	
cnt	RTC counter value (0-0xFFFF FFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */

rtc_lwoff_wait();

/* set counter value to 0xFFFF */

rtc_counter_set(0xFFFF);
```

rtc_prescaler_set

The description of rtc_prescaler_set is shown as below:

Table 3-632. Function rtc_prescaler_set

Function name	rtc_prescaler_set
Function prototype	void rtc_prescaler_set(uint32_t psc);
Function descriptions	set RTC prescaler value
Precondition	before using this function, you must call rtc_lwoff_wait() function (wait until LWOFF flag is set).
The called functions	rtc_configuration_mode_enter / rtc_configuration_mode_exit
Input parameter{in}	
psc	RTC prescaler value (0-0x000F FFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */

rtc_lwoff_wait();

/* set RTC prescaler value to 0x7FFFF */

rtc_prescaler_set(0x7FFFF);
```

rtc_alarm_config

The description of rtc_alarm_config is shown as below:

Table 3-633. Function rtc_alarm_config

Function name	rtc_alarm_config
Function prototype	void rtc_alarm_config(uint32_t alarm);
Function descriptions	set RTC alarm value
Precondition	before using this function, you must call rtc_lwoff_wait() function (wait until LWOFF flag is set).
The called functions	rtc_configuration_mode_enter / rtc_configuration_mode_exit

Input parameter{in}	
alarm	RTC alarm value (0-0xFFFF FFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait();
```

```
/* set alarm value to 0xFFFF */
```

```
rtc_alarm_config(0xFFFF);
```

rtc_divider_get

The description of rtc_divider_get is shown as below:

Table 3-634. Function rtc_divider_get

Function name	rtc_divider_get
Function prototype	uint32_t rtc_divider_get(void);
Function descriptions	get RTC divider value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the value of RTC divider

Example:

```
/* get the current RTC divider value */
```

```
uint32_t rtc_divider_value;
```

```
rtc_divider_value = rtc_divider_get();
```

rtc_flag_get

The description of rtc_flag_getrtc_interrupt_enable is shown as below:

Table 3-635. Function rtc_flag_get

Function name	rtc_flag_get
Function prototype	FlagStatus rtc_flag_get(uint32_t flag);

Function descriptions	get RTC flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	specify which RTC flag status to get
<i>RTC_FLAG_SECOND</i>	second interrupt flag
<i>RTC_FLAG_ALARM</i>	alarm interrupt flag
<i>RTC_FLAG_OVERFLOW</i> <i>W</i>	overflow interrupt flag
<i>RTC_FLAG_RSYN</i>	registers synchronized flag
<i>RTC_FLAG_LWOF</i>	last write operation finished flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the RTC alarm flag status */
```

```
FlagStatus alarm_status;
```

```
alarm_status = rtc_flag_get(RTC_FLAG_ALARM);
```

rtc_flag_clear

The description of rtc_flag_clear is shown as below:

Table 3-636. Function rtc_flag_clear

Function name	rtc_flag_clear
Function prototype	void rtc_flag_clear(uint32_t flag);
Function descriptions	clear RTC flag status
Precondition	before using this function, you must call rtc_lwoff_wait() function (wait until LWOFF flag is set).
The called functions	-
Input parameter{in}	
flag	specify which RTC flag status to clear
<i>RTC_FLAG_SECOND</i>	second interrupt flag
<i>RTC_FLAG_ALARM</i>	alarm interrupt flag
<i>RTC_FLAG_OVERFLOW</i> <i>W</i>	overflow interrupt flag
<i>RTC_FLAG_RSYN</i>	registers synchronized flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */

rtc_lwoff_wait();

/* clear the RTC alarm flag */

rtc_flag_clear(RTC_FLAG_ALARM);
```

rtc_interrupt_enable

The description of rtc_interrupt_enable is shown as below:

Table 3-637. Function rtc_interrupt_enable

Function name	rtc_interrupt_enable
Function prototype	void rtc_interrupt_enable(uint32_t interrupt);
Function descriptions	enable RTC interrupt
Precondition	before using this function, you must call rtc_lwoff_wait() function (wait until LWOFF flag is set).
The called functions	-
Input parameter{in}	
interrupt	specify which RTC interrupt to enable
<i>RTC_INT_SECOND</i>	second interrupt
<i>RTC_INT_ALARM</i>	alarm interrupt
<i>RTC_INT_OVERFLOW</i>	overflow interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */

rtc_lwoff_wait();

/* enable the RTC second interrupt */

rtc_interrupt_enable(RTC_INT_SECOND);
```

rtc_interrupt_disable

The description of rtc_interrupt_disable is shown as below:

Table 3-638. Function rtc_interrupt_disable

Function name	rtc_interrupt_disable
Function prototype	void rtc_interrupt_disable(uint32_t interrupt);
Function descriptions	disable RTC interrupt

Precondition	before using this function, you must call <code>rtc_lwoff_wait()</code> function (wait until LWOFF flag is set).
The called functions	-
Input parameter{in}	
interrupt	specify which RTC interrupt to disable
<code>RTC_INT_SECOND</code>	second interrupt
<code>RTC_INT_ALARM</code>	alarm interrupt
<code>RTC_INT_OVERFLOW</code>	overflow interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait();
```

```
/* disable the RTC second interrupt */
```

```
rtc_interrupt_disable(RTC_INT_SECOND);
```

3.23. SDIO

The secure digital input/output interface (SDIO) defines the SD, SD I/O, MMC and CE-ATA card host interface, which provides command/data transfer between the AHB 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 3-639. SDIO Registers

Registers	Descriptions
SDIO_PWRCTL	SDIO power control register
SDIO_CLKCTL	SDIO clock control register
SDIO_CMDAGMT	SDIO command argument register
SDIO_CMDCTL	SDIO command control register
SDIO_RSPCMDIDX	SDIO command index response register
SDIO_RESPx x=0..3	SDIO response register
SDIO_DATATO	SDIO data timeout register

Registers	Descriptions
SDIO_DATALEN	SDIO data length register
SDIO_DATACTL	SDIO data control register
SDIO_DATACNT	SDIO data counter register
SDIO_STAT	SDIO status register
SDIO_INTC	SDIO interrupt clear register
SDIO_INTEN	SDIO interrupt enable register
SDIO_FIFOCNT	SDIO FIFO counter register
SDIO_FIFO	SDIO FIFO data register

3.23.2. Descriptions of Peripheral functions

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

Table 3-640. 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
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

Function name	Function description
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_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
sdio_interrupt_flag_clear	clear the interrupt pending flags of SDIO

sdio_deinit

The description of sdio_deinit is shown as below:

Table 3-641. Function sdio_deinit

Function name	sdio_deinit
Function prototype	void sdio_deinit(void);
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 3-642. Function sdio_clock_config

Function name	sdio_clock_config
Function prototype	void sdio_clock_config(uint32_t clock_edge, uint32_t clock_bypass, uint32_t clock_powersave, uint16_t clock_division);
Function descriptions	configure the SDIO clock
Precondition	-
The called functions	-
Input parameter{in}	
clock_edge	SDIO_CLK clock edge
SDIO_SDIOLCKEDGE_RISING	select the rising edge of the SDIOCLK to generate SDIO_CLK
SDIO_SDIOLCKEDGE_FALLING	select the falling edge of the SDIOCLK to generate SDIO_CLK
Input parameter{in}	
clock_bypass	clock bypass
SDIO_CLOCKBYPASS_ENABLE	clock bypass
SDIO_CLOCKBYPASS_DISABLE	no bypass
Input parameter{in}	
clock_powersave	SDIO_CLK clock dynamic switch on/off for power saving
SDIO_CLOCKPWRSA_VE_ENABLE	SDIO_CLK closed when bus is idle
SDIO_CLOCKPWRSA_VE_DISABLE	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_SDIOLCKEDGE_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 3-643. 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 3-644. 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 3-645. 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
SDIO_BUSMODE_1BIT T	1-bit SDIO card bus mode
SDIO_BUSMODE_4BIT T	4-bit SDIO card bus mode
SDIO_BUSMODE_8BIT T	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 3-646. Function sdio_power_state_set

Function name	sdio_power_state_set
Function prototype	void sdio_power_state_set(uint32_t power_state);
Function descriptions	set the SDIO power state
Precondition	-
The called functions	-
Input parameter{in}	
power_state	SDIO power state
SDIO_POWER_ON	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 3-647. Function sdio_power_state_get

Function name	sdio_power_state_get
Function prototype	uint32_t sdio_power_state_get(void);
Function descriptions	get the SDIO power state
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	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 3-648. 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 3-649. 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 3-650. 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_RESPONSETYPE_NO</i>	no response
<i>SDIO_RESPONSETYPE_SHORT</i>	short response
<i>SDIO_RESPONSETYPE_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 3-651. Function `sdio_wait_type_set`

Function name	<code>sdio_wait_type_set</code>
Function prototype	<code>void sdio_wait_type_set(uint32_t wait_type);</code>
Function descriptions	set the command state machine wait type
Precondition	-
The called functions	-
Input parameter{in}	
wait_type	wait type
<i>SDIO_WAITTYPE_NO</i>	not wait interrupt
<i>SDIO_WAITTYPE_INTERRUPT</i>	wait interrupt
<i>SDIO_WAITTYPE_DATAEND</i>	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 3-652. 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 3-653. 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 3-654. 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_commond_value;
sdio_commond_value = sdio_command_index_get();
```

sdio_response_get

The description of sdio_response_get is shown as below:

Table 3-655. Function sdio_response_get

Function name	sdio_response_get
Function prototype	uint32_t sdio_response_get(uint32_t response);
Function descriptions	get the response for the last received command
Precondition	-
The called functions	-
Input parameter{in}	
response	SDIO response
SDIO_RESPONSE0	card response[31:0]/card response[127:96]
SDIO_RESPONSE1	card response[95:64]
SDIO_RESPONSE2	card response[63:32]
SDIO_RESPONSE3	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[4] = {0, 0, 0, 0};
```

```
sdio_cid[0] = sdio_response_get(SDIO_RESPONSE0);
```

sdio_data_config

The description of sdio_data_config is shown as below:

Table 3-656. Function sdio_data_config

Function name	sdio_data_config
Function prototype	void sdio_data_config(uint32_t data_timeout, uint32_t data_length, uint32_t data_blocksize);
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
SDIO_DATABLOCKSIZE_1BYTE	block size = 1 byte
SDIO_DATABLOCKSIZE_2BYTES	block size = 2 bytes
SDIO_DATABLOCKSIZE_4BYTES	block size = 4 bytes
SDIO_DATABLOCKSIZE_8BYTES	block size = 8 bytes
SDIO_DATABLOCKSIZE_16BYTES	block size = 16 bytes
SDIO_DATABLOCKSIZE_32BYTES	block size = 32 bytes
SDIO_DATABLOCKSIZE_64BYTES	block size = 64 bytes
SDIO_DATABLOCKSIZE_128BYTES	block size = 128 bytes
SDIO_DATABLOCKSIZE_256BYTES	block size = 256 bytes
SDIO_DATABLOCKSIZE_512BYTES	block size = 512 bytes
SDIO_DATABLOCKSIZE_1024BYTES	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 3-657. 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
<i>SDIO_TRANSMODE_BLOCK</i>	block transfer
<i>SDIO_TRANSMODE_STREAM</i>	stream transfer or SDIO multibyte transfer
Input parameter{in}	
transfer_direction	data transfer direction, read or write
<i>SDIO_TRANSDIRECTION_TOCARD</i>	write data to card
<i>SDIO_TRANSDIRECTION_TOSDIO</i>	read data from card
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure SDIO data transmisson */

sdio_data_transfer_config(SDIO_TRANSDIRECTION_TOSDIO,
SDIO_TRANSMODE_BLOCK);
```

sdio_dsm_enable

The description of sdio_dsm_enable is shown as below:

Table 3-658. 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 3-659. 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 3-660. 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 3-661. Function sdio_data_read

Function name	sdio_data_read
Function prototype	uint32_t sdio_data_read(void);
Function descriptions	read data(one word) from the receive FIFO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	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 3-662. Function sdio_data_counter_get

Function name	sdio_data_counter_get
Function prototype	uint32_t sdio_data_counter_get(void);
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 3-663. 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 3-664. 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 3-665. 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_readwait_enable

The description of sdio_readwait_enable is shown as below:

Table 3-666. Function sdio_readwait_enable

Function name	sdio_readwait_enable
Function prototype	void sdio_readwait_enable(void);
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 3-667. Function sdio_readwait_disable

Function name	sdio_readwait_disable
Function prototype	void sdio_readwait_disable(void);
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 3-668. Function sdio_stop_readwait_enable

Function name	sdio_stop_readwait_enable
Function prototype	void sdio_stop_readwait_enable(void);
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 3-669. 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 3-670. 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
SDIO_READWAITTYPE_CLK	read wait control by stopping SDIO_CLK
SDIO_READWAITTYPE_DAT2	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(SDIO_READWAITTYPE_CLK);
```

sdio_operation_enable

The description of sdio_operation_enable is shown as below:

Table 3-671. 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 3-672. 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 3-673. 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 3-674. 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 3-675. Function sdio_ceata_command_enable

Function name	sdio_ceata_command_enable
Function prototype	void sdio_ceata_command_enable(void);
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 3-676. Function sdio_ceata_command_disable

Function name	sdio_ceata_command_disable
Function prototype	void sdio_ceata_command_disable(void);
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 3-677. 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 3-678. 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 3-679. 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 3-680. 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();
```

sdio_flag_get

The description of sdio_flag_get is shown as below:

Table 3-681. 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_CCRCE</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>	command response received (CRC check passed) flag

V	
SDIO_FLAG_CMDSEN	command sent (no response required) flag
D	
SDIO_FLAG_DTEND	data end (data counter, SDIO_DATACNT, is zero) flag
SDIO_FLAG_STBITE	start bit error in the bus flag
SDIO_FLAG_DTBLKE	data block sent/received (CRC check passed) flag
ND	
SDIO_FLAG_CMDRUN	command transmission in progress flag
SDIO_FLAG_TXRUN	data transmission in progress flag
SDIO_FLAG_RXRUN	data reception in progress flag
SDIO_FLAG_TFH	transmit FIFO is half empty flag: at least 8 words can be written into the FIFO
SDIO_FLAG_RFH	receive FIFO is half full flag: at least 8 words can be read in the FIFO
SDIO_FLAG_TFF	transmit FIFO is full flag
SDIO_FLAG_RFF	receive FIFO is full flag
SDIO_FLAG_TFE	transmit FIFO is empty flag
SDIO_FLAG_RFE	receive FIFO is empty flag
SDIO_FLAG_TXDTVAL	data is valid in transmit FIFO flag
SDIO_FLAG_RXDTVAL	data is valid in receive FIFO flag
L	
SDIO_FLAG_SDIOINT	SD I/O interrupt received flag
SDIO_FLAG_ATAEND	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 3-682. Function sdio_flag_clear

Function name	sdio_flag_clear
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_CCR CER</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_DATA CNT, 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 3-683. 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}	
int_flag	interrupt flags state of SDIO
<i>SDIO_INT_CCR CER</i>	SDIO CCR CER interrupt

<i>SDIO_INT_DTCRCER</i> <i>R</i>	SDIO DTCRCERR interrupt
<i>SDIO_INT_CMDTMOU</i> <i>T</i>	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:

```
/* enable the SDIO corresponding interrupts */
```

```
sdio_interrupt_enable(SDIO_INT_CCRERR | 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 3-684. Function sdio_interrupt_disable

Function name	sdio_interrupt_disable
Function prototype	void sdio_interrupt_disable(uint32_t int_flag);
Function descriptions	disable the SDIO interrupt

Precondition	-
The called functions	-
Input parameter{in}	
int_flag	interrupt flags state of SDIO
<i>SDIO_INT_CCRRCERR</i>	SDIO CCRRCERR interrupt
<i>SDIO_INT_DTCRCERR</i>	SDIO DTCRCERR interrupt
<i>SDIO_INT_CMDTMOUT</i>	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 3-685. 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
<code>SDIO_INT_FLAG_DTC RCERR</code>	SDIO DTCRCERR interrupt
<code>SDIO_INT_FLAG_CMD TMOUT</code>	SDIO CMDTMOUT interrupt
<code>SDIO_INT_FLAG_DTT MOUT</code>	SDIO DTTMOUT interrupt
<code>SDIO_INT_FLAG_TXU RE</code>	SDIO TXURE interrupt
<code>SDIO_INT_FLAG_RXO RE</code>	SDIO_INT_RXORE
<code>SDIO_INT_FLAG_CMD RECV</code>	SDIO CMDRECV interrupt
<code>SDIO_INT_FLAG_CMD SEND</code>	SDIO CMDSEND interrupt
<code>SDIO_INT_FLAG_DTE ND</code>	SDIO DTEND interrupt
<code>SDIO_INT_FLAG_STBI TE</code>	SDIO STBITE interrupt
<code>SDIO_INT_FLAG_DTB LKEND</code>	SDIO DTBLKEND interrupt
<code>SDIO_INT_FLAG_CMD RUN</code>	SDIO CMDRUN interrupt
<code>SDIO_INT_FLAG_TXR UN</code>	SDIO TXRUN interrupt
<code>SDIO_INT_FLAG_RXR UN</code>	SDIO RXRUN interrupt
<code>SDIO_INT_FLAG_TFH</code>	SDIO TFH interrupt
<code>SDIO_INT_FLAG_RFH</code>	SDIO RFH interrupt
<code>SDIO_INT_FLAG_TFF</code>	SDIO TFF interrupt
<code>SDIO_INT_FLAG_RFF</code>	SDIO RFF interrupt
<code>SDIO_INT_FLAG_TFE</code>	SDIO TFE interrupt
<code>SDIO_INT_FLAG_RFE</code>	SDIO RFE interrupt
<code>SDIO_INT_FLAG_TXD</code>	SDIO TXDTVALL interrupt

<i>TVAL</i>	
<i>SDIO_INT_FLAG_RXD TVAL</i>	SDIO RXDTVAl interrupt
<i>SDIO_INT_FLAG_SDI OINT</i>	SDIO SDIOINT interrupt
<i>SDIO_INT_FLAG_ATA END</i>	SDIO ATAEND interrupt
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 3-686. Function sdio_interrupt_flag_clear

Function name	sdio_interrupt_flag_clear
Function prototype	void sdio_interrupt_flag_clear(uint32_t int_flag);
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 CERR</i>	SDIO CCRCERR interrupt
<i>SDIO_INT_FLAG_DTC RCERR</i>	SDIO DTCRCERR interrupt
<i>SDIO_INT_FLAG_CMD TMOUT</i>	SDIO CMDTMOUT interrupt
<i>SDIO_INT_FLAG_DTT MOUT</i>	SDIO DTTMOUT interrupt
<i>SDIO_INT_FLAG_TXU RE</i>	SDIO TXURE interrupt
<i>SDIO_INT_FLAG_RXO RE</i>	SDIO_INT_RXORE
<i>SDIO_INT_FLAG_CMD RECV</i>	SDIO CMDRECV interrupt

<i>SDIO_INT_FLAG_CMD SEND</i>	SDIO CMDSEND interrupt
<i>SDIO_INT_FLAG_DTE ND</i>	SDIO DTEND interrupt
<i>SDIO_INT_FLAG_STBI TE</i>	SDIO STBITE interrupt
<i>SDIO_INT_FLAG_DTB LKEND</i>	SDIO DTBLKEND interrupt
<i>SDIO_INT_FLAG_SDI OINT</i>	SDIO SDIOINT interrupt
<i>SDIO_INT_FLAG_ATA END</i>	SDIO ATAEND interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the interrupt pending flags of SDIO */
```

```
sdio_interrupt_flag_clear(SDIO_INT_FLAG_DTEND);
```

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 3-687. SPI/I2S Registers

Registers	Descriptions
SPI_CTL0	SPI control register 0
SPI_CTL1	SPI control register 1
SPI_STAT	SPI status register
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

Registers	Descriptions
SPI_QCTL	Quad-SPI mode control register

3.24.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

Table 3-688. SPI/I2S firmware function

Function name	Function description
spi_i2s_deinit	reset SPI and I2S
spi_struct_para_init	initialize the parameters of SPI structure with the default values
spi_init	initialize SPI parameters
spi_enable	enable SPI
spi_disable	disable SPI
i2s_init	initialize I2S parameters
i2s_psc_config	configure I2S prescaler
i2s_enable	enable I2S
i2s_disable	disable I2S
spi_nss_output_enable	enable SPI NSS output
spi_nss_output_disable	disable SPI NSS output
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 send or receive
spi_dma_disable	disable SPI DMA send or receive
spi_i2s_data_frame_format_config	configure SPI data frame format
spi_bidirectional_transfer_config	configure SPI bidirectional transfer direction
spi_i2s_data_transmit	SPI transmit data
spi_i2s_data_receive	SPI receive data
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_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

Function name	Function description
spi_i2s_interrupt_disable	disable SPI and I2S interrupt
spi_i2s_interrupt_flag_get	get SPI and I2S interrupt status
spi_crc_error_clear	clear SPI CRC error flag status

Structure spi_parameter_struct

Table 3-689. Structure spi_parameter_struct

Member name	Function description
device_mode	SPI master or slave (SPI_MASTER, SPI_SLAVE)
trans_mode	SPI transfer type (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	SPI frame size (SPI_FRAME_SIZE_16BIT, SPI_FRAME_SIZE_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_CLOCK_PHASE_LOW, SPI_CLOCK_PHASE_HIGH, SPI_CLOCK_PHASE_LOW_2EDGE, SPI_CLOCK_PHASE_HIGH_2EDGE)
prescale	SPI prescaler 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 3-690. 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
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
spi_periph	SPI/I2S peripheral
SP/x(x=0,1,2)	SPI/I2S peripheral selection
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 3-691. 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 structure with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
spi_struct	SPI parameter struct, the structure members can refer to members of the structure 错误!未找到引用源。
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 3-692. 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 parameter
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx(x=0,1,2)	SPI peripheral selection
Input parameter{in}	
spi_struct	SPI parameter initialization structure, the structure members can refer to members of the structure 错误!未找到引用源。

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_FRAME_SIZE_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 3-693. 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)	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable SPI0 */

spi_enable(SPI0);

```

spi_disable

The description of spi_disable is shown as below:

Table 3-694. 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
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 */
spi_disable(SPI0);
```

i2s_init

The description of i2s_init is shown as below:

Table 3-695. Function i2s_init

Function name	i2s_init
Function prototype	void i2s_init(uint32_t spi_periph, uint32_t mode, uint32_t standard, uint32_t ckpl);
Function descriptions	initialize I2S parameter
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	I2S peripheral
<i>SPIx(x=1,2)</i>	I2S peripheral selection
Input parameter{in}	
mode	I2S operation mode
<i>I2S_MODE_SLAVETX</i>	I2S slave transmit mode
<i>I2S_MODE_SLAVRX</i>	I2S slave receive mode
<i>I2S_MODE_MASTERTX</i>	I2S master transmit mode
<i>I2S_MODE_MASTERRX</i>	I2S master receive mode

Input parameter{in}	
standard	I2S standard
<i>I2S_STD_PHILIPS</i>	I2S philips 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}	
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_PHILIPS, I2S_CKPL_LOW);
```

i2s_psc_config

The description of i2s_psc_config is shown as below:

Table 3-696. Function i2s_psc_config

Function name	i2s_psc_config
Function prototype	void i2s_psc_config(uint32_t spi_periph, uint32_t audiosample, uint32_t frameformat, uint32_t mckout);
Function descriptions	configure I2S prescaler
Precondition	-
The called functions	rcu_clock_freq_get
Input parameter{in}	
spi_periph	I2S peripheral
<i>SPIx(x=1,2)</i>	I2S peripheral selection
Input parameter{in}	
audiosample	I2S audio sample rate
<i>I2S_AUDIOSAMPLE_8K</i>	audio sample rate is 8KHz
<i>I2S_AUDIOSAMPLE_11K</i>	audio sample rate is 11KHz
<i>I2S_AUDIOSAMPLE_16K</i>	audio sample rate is 16KHz
<i>I2S_AUDIOSAMPLE_22K</i>	audio sample rate is 22KHz

2K	
I2S_AUDIOSAMPLE_3 2K	audio sample rate is 32KHz
I2S_AUDIOSAMPLE_4 4K	audio sample rate is 44KHz
I2S_AUDIOSAMPLE_4 8K	audio sample rate is 48KHz
I2S_AUDIOSAMPLE_9 6K	audio sample rate is 96KHz
I2S_AUDIOSAMPLE_1 92K	audio sample rate is 192KHz
Input parameter{in}	
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}	
mckout	I2S master clock output
I2S_MCKOUT_ENABL E	I2S master clock output enable
I2S_MCKOUT_DISABL E	I2S master clock output disable
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 3-697. 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)	I2S peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2S1*/
i2s_enable(SPI1);
```

i2s_disable

The description of i2s_disable is shown as below:

Table 3-698. 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)	I2S peripheral selection
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 3-699. 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
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
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 3-700. 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
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
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 3-701. 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
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
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 3-702. 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
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
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 3-703. 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 send or receive
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
Input parameter{in}	
dma	SPI DMA mode
<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA
<i>SPI_DMA_RECEIVE</i>	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 3-704. 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 send or receive
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
Input parameter{in}	
dma	SPI DMA mode
<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA
<i>SPI_DMA_RECEIVE</i>	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 3-705. 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 data frame format
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
Input parameter{in}	
frame_format	SPI frame size
<i>SPI_FRAME_SIZE_16BIT</i>	SPI frame size is 16 bits
<i>SPI_FRAME_SIZE_8BIT</i>	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_FRAME_SIZE_16BIT);
```

spi_bidirectional_transfer_config

The description of spi_bidirectional_transfer_config is shown as below:

Table 3-706. 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(x=0,1,2)</i>	SPI peripheral selection
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);
```

spi_i2s_data_transmit

The description of spi_i2s_data_transmit is shown as below:

Table 3-707. 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
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
Input parameter{in}	
data	16-bit data
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 transmit data */
```

```
uint16_t spi0_send_array[10];
```

```
uint8_t send_n = 1;
```

```
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n]);
```

spi_i2s_data_receive

The description of spi_i2s_data_receive is shown as below:

Table 3-708. 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(x=0,1,2)</i>	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
uint16_t	16-bit data

Example:

```
/* SPI0 receive data */
uint16_t spi0_receive_array[10];
uint8_t receive_n = 1;
spi0_receive_array[receive_n] = spi_i2s_data_receive(SPI0);
```

spi_crc_polynomial_set

The description of spi_crc_polynomial_set is shown as below:

Table 3-709. 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
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
Input parameter{in}	
crc_poly	CRC polynomial value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set SPI0 CRC polynomial */

uint16_t CRC_VALUE = 0x8;

spi_crc_polynomial_set(SPI0,CRC_VALUE);
```

spi_crc_polynomial_get

The description of spi_crc_polynomial_get is shown as below:

Table 3-710. 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
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
uint16_t	16-bit CRC polynomial (0-0xFFFF)

Example:

```
/* get SPI0 CRC polynomial */

uint16_t CRC_VALUE;

CRC_VALUE = spi_crc_polynomial_get(SPI0);
```

spi_crc_on

The description of spi_crc_on is shown as below:

Table 3-711. Function spi_crc_on

Function name	spi_crc_on
Function prototype	void spi_crc_on(uint32_t spi_periph);
Function descriptions	turn on CRC function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
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 3-712. Function spi_crc_off

Function name	spi_crc_off
Function prototype	void spi_crc_off(uint32_t spi_periph);
Function descriptions	turn off CRC function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
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 3-713. 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
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* SPI0 next data is CRC value */
```

```
spl_crc_next(SPI0);
```

spl_crc_get

The description of spl_crc_get is shown as below:

Table 3-714. Function spl_crc_get

Function name	spl_crc_get
Function prototype	uint16_t spl_crc_get(uint32_t spl_periph, uint8_t crc);
Function descriptions	get SPI CRC send value or receive value
Precondition	-
The called functions	-
Input parameter{in}	
spl_periph	SPI peripheral
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
Input parameter{in}	
crc	SPI crc value
<i>SPI_CRC_TX</i>	get transmit crc value
<i>SPI_CRC_RX</i>	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 value;
```

```
value = spl_crc_get(SPI0, SPI_CRC_TX);
```

spl_quad_enable

The description of spl_quad_enable is shown as below:

Table 3-715. Function spl_quad_enable

Function name	spl_quad_enable
Function prototype	void spl_quad_enable(uint32_t spl_periph);
Function descriptions	enable quad wire SPI

Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx(x=0)</i>	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable quad wire SPI */
```

```
spi_quad_enable(SPI0);
```

spi_quad_disable

The description of spi_quad_disable is shown as below:

Table 3-716. Function spi_quad_enable

Function name	spi_quad_disable
Function prototype	void spi_quad_disable(uint32_t spi_periph);
Function descriptions	disable quad wire SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx(x=0)</i>	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable quad wire SPI */
```

```
spi_quad_disable(SPI0);
```

spi_quad_write_enable

The description of spi_quad_write_enable is shown as below:

Table 3-717. 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
<i>SPIx(x=0)</i>	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable quad wire SPI write */
```

```
spi_quad_write_enable(SPI0);
```

spi_quad_read_enable

The description of spi_quad_read_enable is shown as below:

Table 3-718. 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
<i>SPIx(x=0)</i>	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable quad wire SPI read */
```

```
spi_quad_read_enable(SPI0);
```

spi_quad_io23_output_enable

The description of spi_quad_io23_output_enable is shown as below:

Table 3-719. 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
<i>SPIx(x=0)</i>	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_enable(SPI0);
```

spi_quad_io23_output_disable

The description of spi_quad_io23_output_disable is shown as below:

Table 3-720. 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
<i>SPIx</i>	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_disable(SPI0);
```

spi_i2s_flag_get

The description of spi_i2s_flag_get is shown as below:

Table 3-721. Function spi_i2s_flag_get

Function name	spi_i2s_flag_get
Function prototype	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);
Function descriptions	get SPI and I2S flag status

Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI/I2S peripheral
<i>SPIx(x=0,1,2)</i>	SPI/I2S peripheral selection
Input parameter{in}	
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_I2S_INT_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>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
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty flag status */
```

```
FlagStatus Flag = RESET;
```

```
Flag = spi_i2s_flag_get(SPI0, SPI_FLAG_TBE);
```

spi_i2s_interrupt_enable

The description of spi_i2s_interrupt_enable is shown as below:

Table 3-722. Function spi_i2s_interrupt_enable

Function name	spi_i2s_interrupt_enable
Function prototype	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);
Function descriptions	enable SPI and I2S interrupt
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI/I2S peripheral
<i>SPIx(x=0,1,2)</i>	SPI/I2S peripheral selection

Input parameter{in}	
interrupt	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 3-723. Function spi_i2s_interrupt_disable

Function name	spi_i2s_interrupt_disable
Function prototype	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
Function descriptions	disable SPI and I2S interrupt
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI/I2S peripheral
<i>SPIx(x=0,1,2)</i>	SPI/I2S peripheral selection
Input parameter{in}	
interrupt	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 3-724. 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 interrupt);
Function descriptions	get SPI and I2S interrupt status
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI/I2S peripheral
<i>SPIx(x=0,1,2)</i>	SPI/I2S peripheral selection
Input parameter{in}	
interrupt	SPI/I2S interrupt flag status
<i>SPI_I2S_INT_FLAG_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_FLAG_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_FLAG_RXORERR</i>	overrun interrupt
<i>SPI_INT_FLAG_CONFERR</i>	config error interrupt
<i>SPI_INT_FLAG_CRCERR</i>	CRC error interrupt
<i>I2S_INT_FLAG_TXURERR</i>	underrun error interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty interrupt status */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE);
```

spi_crc_error_clear

The description of spi_crc_error_clear is shown as below:

Table 3-725. 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
<i>SPIx(x=0,1,2)</i>	SPI peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear SPI0 CRC error flag status */
```

```
spi_crc_error_clear(SPI0);
```

3.25. 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), and basic timer (TIMERx, x=5, 6). The specific functions of different types of timer are different. The TIMER 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

TIMERx registers are listed in the table shown as below:

Table 3-726. TIMERx Registers

Registers	Descriptions
TIMER_CTL0	control register 0
TIMER_CTL1	control register 1
TIMER_SMCFG	slave mode configuration register
TIMER_DMAINTEN	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

Registers	Descriptions
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

3.25.2. Descriptions of Peripheral functions

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

Table 3-727. TIMEx firmware function

Function name	Function description
timer_deinit	deinit a TIMER
timer_struct_para_init	initialize TIMER init parameter struct
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	configure TIMER counter alignment mode
timer_counter_up_direction	configure TIMER counter up direction
timer_counter_down_direction	configure 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
timer_dma_transfer_config	configure the TIMER DMA transfer
timer_event_software_generate	software generate events

Function name	Function description
timer_break_struct_para_init	initialize TIMER break parameter struct
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	enable or disable TIMER primary output function
timer_channel_control_shadow_config	enable or disable channel capture/compare control shadow register
timer_channel_control_shadow_update_config	configure TIMER channel control shadow register update control
timer_channel_output_struct_para_init	initialize TIMER channel output parameter struct
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_state_config	configure TIMER channel complementary output enable state
timer_channel_input_struct_para_init	initialize TIMER channel input parameter struct
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

Function name	Function description
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_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 flags
timer_interrupt_flag_clear	clear TIMER interrupt flags

Structure timer_parameter_struct

Table 3-728. 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 3-729. 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)
protectmode	complementary register protect control(TIMER_CCHP_PROT_OFF, TIMER_CCHP_PROT_0, TIMER_CCHP_PROT_1, TIMER_CCHP_PROT_2)

Member name	Function description
breakstate	break enable(TIMER_BREAK_ENABLE, TIMER_BREAK_DISABLE)

Structure timer_oc_parameter_struct

Table 3-730. 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 3-731. 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 3-732. 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
TIMERx(x=0..13)	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 3-733. 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 TIMER init parameter struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
initpara	TIMER init parameter struct, the structure members can refer to Table 3-728. Structure timer_parameter_struct .
Return value	
-	-

Example:

```
/* initialize the TIMER structure */
```

```
timer_parameter_struct initpara;
```

```
timer_struct_para_init(&initpara);
```

timer_init

The description of timer_init is shown as below:

Table 3-734. 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 3-728. 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 3-735. 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(TIMERO);
```

timer_disable

The description of timer_disable is shown as below:

Table 3-736. 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 TIMERO */
```

```
timer_disable(TIMERO);
```

timer_auto_reload_shadow_enable

The description of timer_auto_reload_shadow_enable is shown as below:

Table 3-737. 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 TIMERO auto reload shadow function */
```

timer_auto_reload_shadow_enable(TIMERO);

timer_auto_reload_shadow_disable

The description of timer_auto_reload_shadow_disable is shown as below:

Table 3-738. 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
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMERO auto reload shadow function */
```

```
timer_auto_reload_shadow_disable(TIMERO);
```

timer_update_event_enable

The description of timer_update_event_enable is shown as below:

Table 3-739. 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 TIMERO the update event */
```

```
timer_update_event_enable(TIMER0);
```

timer_update_event_disable

The description of timer_update_event_disable is shown as below:

Table 3-740. 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 3-741. Function timer_counter_alignment

Function name	timer_counter_alignment
Function prototype	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);
Function descriptions	configure 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_COUNTER_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 <i>TIMERx_CHCTL0</i> register). 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_CHCTL0 register). 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 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 3-742. 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	configure 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 3-743. 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	configure TIMER counter no center-aligned mode(edge-aligned mode)
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..4,7..13)	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 3-744. 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
TIMERx(x=0..13)	TIMER peripheral selection
Input parameter{in}	
prescaler	prescaler value(0~65535)
Input parameter{in}	
pscreload	prescaler reload mode
TIMER_PSC_RELOAD_NOW	the prescaler is loaded right now
TIMER_PSC_RELOAD_UPDATE	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 3-745. Function timer_repetition_value_config

Function name	timer_repetition_value_config
Function prototype	void timer_repetition_value_config(uint32_t timer_periph, uint8_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 3-746. 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, 0~65535
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 3-747. 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, 0~65535
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 3-748. 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 3-749. 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, 0~65535

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 3-750. 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..11)</i>	TIMER peripheral selection
Input parameter{in}	
spmode	pulse mode
<i>TIMER_SP_MODE_SINGLE</i>	single pulse mode
<i>TIMER_SP_MODE_REPETITIVE</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 3-751. Function timer_update_source_config

Function name	timer_update_source_config
Function prototype	void timer_update_source_config(uint32_t timer_periph, uint8_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: <ul style="list-style-type: none"> - 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_</i>	only counter overflow/underflow generates an update interrupt or DMA

REGULAR	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 3-752. 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
TIMERx	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
dma	TIMER DMA source enable
TIMER_DMA_UPD	update DMA enable, TIMERx(x=0..7)
TIMER_DMA_CH0D	channel 0 DMA enable, TIMERx(x=0..4, 7)
TIMER_DMA_CH1D	channel 1 DMA enable, TIMERx(x=0..4, 7)
TIMER_DMA_CH2D	channel 2 DMA enable, TIMERx(x=0..4, 7)
TIMER_DMA_CH3D	channel 3 DMA enable, TIMERx(x=0..4, 7)
TIMER_DMA_CMTD	commutation DMA request enable, TIMERx(x=0, 7)
TIMER_DMA_TRGD	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 3-753. 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>	TIMER peripheral selection (x 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 3-754. 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, uint8_t dma_request);
Function descriptions	channel DMA request source selection
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral

<i>TIMERx(x=0..4, 7)</i>	TIMER peripheral selection
Input parameter{in}	
dma_request	channel DMA request source selection
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	DMA request of channel y is sent when channel y event occurs
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	DMA request of channel y is sent when update event occurs
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* TIMER0 channel DMA request of channel y 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 3-755. 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>	TIMER peripheral selection (x 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, <i>TIMERx(x=0..4, 7)</i>
<i>TIMER_DMACFG_DMA_TA_CTL1</i>	DMA transfer address is TIMER_CTL1, <i>TIMERx(x=0..4, 7)</i>
<i>TIMER_DMACFG_DMA_TA_SMCFG</i>	DMA transfer address is TIMER_SMCFG, <i>TIMERx(x=0..4, 7)</i>
<i>TIMER_DMACFG_DMA_TA_DMAINTEN</i>	DMA transfer address is TIMER_DMAINTEN, <i>TIMERx(x=0..4, 7)</i>
<i>TIMER_DMACFG_DMA_TA_INTF</i>	DMA transfer address is TIMER_INTF, <i>TIMERx(x=0..4, 7)</i>

<i>TIMER_DMACFG_DMA</i> <i>TA_SWEVG</i>	DMA transfer address is <i>TIMER_SWEVG</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CHCTL0</i>	DMA transfer address is <i>TIMER_CHCTL0</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CHCTL1</i>	DMA transfer address is <i>TIMER_CHCTL1</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CHCTL2</i>	DMA transfer address is <i>TIMER_CHCTL2</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CNT</i>	DMA transfer address is <i>TIMER_CNT</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMA</i> <i>TA_PSC</i>	DMA transfer address is <i>TIMER_PSC</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CAR</i>	MA transfer address is <i>TIMER_CAR</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CREP</i>	DMA transfer address is <i>TIMER_CREP</i> , <i>TIMERx</i> (<i>x</i> =0, 7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH0CV</i>	DMA transfer address is <i>TIMER_CH0CV</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH1CV</i>	DMA transfer address is <i>TIMER_CH1CV</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH2CV</i>	DMA transfer address is <i>TIMER_CH2CV</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH3CV</i>	DMA transfer address is <i>TIMER_CH3CV</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CCHP</i>	DMA transfer address is <i>TIMER_CCHP</i> , <i>TIMERx</i> (<i>x</i> =0, 7)
<i>TIMER_DMACFG_DMA</i> <i>TA_DMACFG</i>	DMA transfer address is <i>TIMER_DMACFG</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMA</i> <i>TA_DMATB</i>	DMA transfer address is <i>TIMER_DMATB</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
Input parameter{in}	
dma_lenth	DMA transfer count, <i>TIMER_DMACFG_DMATC_xTRANSFER</i> (<i>x</i> =1..18)
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 3-756. 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>	TIMER peripheral selection (x 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_TRIGG</i>	trigger event generation, TIMERx(x=0..4, 7, 8, 11)
<i>TIMER_EVENT_SRC_BREAKG</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 3-757. 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 TIMER break parameter struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
breakpara	TIMER break parameter struct, the structure members can refer to Table 3-729. Structure timer break parameter struct.
Return value	
-	-

Example:

```
/* initialize the TIMER break parameter structure */
```

```
timer_break_parameter_struct breakpara;
```

```
timer_break_struct_para_init(&breakpara);
```

timer_break_config

The description of timer_break_config is shown as below:

Table 3-758. 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
TIMERx(x=0, 7)	TIMER peripheral selection
Input parameter{in}	
breakpara	TIMER break parameter struct, the structure members can refer to Table 3-729. Structure timer break parameter struct.
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 3-759. 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-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 break function */

timer_break_enable(TIMER0);
```

timer_break_disable

The description of timer_break_disable is shown as below:

Table 3-760. 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-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 break function */
```

```
timer_break_disable(TIMER0);
```

timer_automatic_output_enable

The description of timer_automatic_output_enable t is shown as below:

Table 3-761. 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 t is shown as below:

Table 3-762. 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 3-763. 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	enable or disable 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
<i>ENABLE</i>	enable function
<i>DISABLE</i>	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 3-764. 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	enable or disable channel capture/compare control shadow register
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 3-765. 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, uint32_t ccuctl);
Function descriptions	configure TIMER channel 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 3-766. 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* ocpa);
Function descriptions	initialize TIMER channel output parameter struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
ocpara	TIMER channel output parameter struct, the structure members can refer to Table 3-730. Structure timer_oc_parameter_struct .
Return value	
-	-

Example:

```
/* initialize the TIMER channel output parameter structure */
timer_oc_parameter_struct ocpa;
timer_channel_output_struct_para_init(&ocpa);
```

timer_channel_output_config

The description of timer_channel_output_config is shown as below:

Table 3-767. 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
<i>TIMERx</i>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel0(TIMERx(x=0..4, 7..13))
<i>TIMER_CH_1</i>	TIMER channel1(TIMERx(x=0..4, 7, 8, 11))
<i>TIMER_CH_2</i>	TIMER channel2(TIMERx(x=0..4, 7))
<i>TIMER_CH_3</i>	TIMER channel3(TIMERx(x=0..4, 7))
Input parameter{in}	
ocpara	TIMER channel output parameter struct, the structure members can refer to Table 3-730. Structure timer_oc_parameter_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output function */
timer_oc_parameter_struct timer_ocintpara;
timer_ocintpara.outputstate = TIMER_CCX_ENABLE;
timer_ocintpara.outputnstate = TIMER_CCXN_ENABLE;
timer_ocintpara.ocpolarity = TIMER_OC_POLARITY_HIGH;
timer_ocintpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;
timer_ocintpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;
timer_ocintpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;
timer_channel_output_config(TIMER0, TIMER_CH_0, &timer_ocintpara);
```

timer_channel_output_mode_config

The description of timer_channel_output_mode_config is shown as below:

Table 3-768. Function timer_channel_output_mode_config

Function name	timer_channel_output_mode_config
Function prototype	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);
Function descriptions	configure TIMER channel output compare mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel0(TIMERx(x=0..4, 7..13))
<i>TIMER_CH_1</i>	TIMER channel1(TIMERx(x=0..4, 7, 8, 11))
<i>TIMER_CH_2</i>	TIMER channel2(TIMERx(x=0..4, 7))
<i>TIMER_CH_3</i>	TIMER channel3(TIMERx(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>	active mode
<i>TIMER_OC_MODE_INACTIVE</i>	inactive mode
<i>TIMER_OC_MODE_TOGGLE</i>	toggle mode
<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 3-769. 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, uint16_t pulse);
Function descriptions	configure TIMER channel output pulse value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel0(TIMERx(x=0..4, 7..13))
<i>TIMER_CH_1</i>	TIMER channel1(TIMERx(x=0..4, 7, 8, 11))
<i>TIMER_CH_2</i>	TIMER channel2(TIMERx(x=0..4, 7))
<i>TIMER_CH_3</i>	TIMER channel3(TIMERx(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 3-770. 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>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel0(<i>TIMERx</i> (<i>x</i> =0..4, 7..13))
<i>TIMER_CH_1</i>	TIMER channel1(<i>TIMERx</i> (<i>x</i> =0..4, 7, 8, 11))
<i>TIMER_CH_2</i>	TIMER channel2(<i>TIMERx</i> (<i>x</i> =0..4, 7))
<i>TIMER_CH_3</i>	TIMER channel3(<i>TIMERx</i> (<i>x</i> =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 3-771. 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>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel0(<i>TIMERx</i> (<i>x</i> =0..4, 7..13))
<i>TIMER_CH_1</i>	TIMER channel1(<i>TIMERx</i> (<i>x</i> =0..4, 7, 8, 11))
<i>TIMER_CH_2</i>	TIMER channel2(<i>TIMERx</i> (<i>x</i> =0..4, 7))
<i>TIMER_CH_3</i>	TIMER channel3(<i>TIMERx</i> (<i>x</i> =0..4, 7))

Input parameter{in}	
ocfast	channel output fast function
<i>TIMER_OC_FAST_ENABLE</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 3-772. 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 peripheral
<i>TIMERx(x=0..4, 7)</i>	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel0
<i>TIMER_CH_1</i>	TIMER channel1
<i>TIMER_CH_2</i>	TIMER channel2
<i>TIMER_CH_3</i>	TIMER channel3
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 3-773. 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
<i>TIMERx</i>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel0(TIMERx(x=0..4, 7..13))
<i>TIMER_CH_1</i>	TIMER channel1(TIMERx(x=0..4, 7, 8, 11))
<i>TIMER_CH_2</i>	TIMER channel2(TIMERx(x=0..4, 7))
<i>TIMER_CH_3</i>	IMER channel3(TIMERx(x=0..4, 7))
Input parameter{in}	
ocpolarity	channel output polarity
<i>TIMER_OC_POLARITY_HIGH</i>	channel output polarity is high
<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 3-774. 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</i>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel0(TIMERx(x=0..4, 7..13))
<i>TIMER_CH_1</i>	TIMER channel1(TIMERx(x=0..4, 7, 8, 11))
<i>TIMER_CH_2</i>	TIMER channel2(TIMERx(x=0..4, 7))
Input parameter{in}	
ocnpolarity	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,
TIMER_OCN_POLARITY_HIGH);
```

timer_channel_output_state_config

The description of timer_channel_output_state_config is shown as below:

Table 3-775. 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>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel0(TIMERx(x=0..4, 7..13))
<i>TIMER_CH_1</i>	TIMER channel1(TIMERx(x=0..4, 7, 8, 11))
<i>TIMER_CH_2</i>	TIMER channel2(TIMERx(x=0..4, 7))
<i>TIMER_CH_3</i>	TIMER channel3(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 3-776. 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 channel0
<i>TIMER_CH_1</i>	TIMER channel1
<i>TIMER_CH_2</i>	TIMER channel2

Input parameter{in}	
ocnstate	TIMER channel complementary output enable state
<i>TIMER_CCXN_ENABLE</i>	channel complementary enable
<i>TIMER_CCXN_DISABL</i> <i>E</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,
TIMER_CCXN_ENABLE);
```

timer_channel_input_struct_para_init

The description of timer_channel_input_struct_para_init is shown as below:

Table 3-777. 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 TIMER channel input parameter struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
icpara	TIMER channel input parameter struct, the structure members can refer to Table 3-731. Structure timer_ic_parameter_struct .
Return value	
-	-

Example:

```
/* initialize the TIMER channel input parameter structure */
timer_ic_parameter_struct icpara;
timer_channel_input_struct_para_init(&icpara);
```

timer_input_capture_config

The description of timer_input_capture_config is shown as below:

Table 3-778. 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
<i>TIMERx</i>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel0(TIMERx(x=0..4, 7..13))
<i>TIMER_CH_1</i>	TIMER channel1(TIMERx(x=0..4, 7, 8, 11))
<i>TIMER_CH_2</i>	TIMER channel2(TIMERx(x=0..4, 7))
<i>TIMER_CH_3</i>	TIMER channel3(TIMERx(x=0..4, 7))
Input parameter{in}	
icpara	TIMER channel input parameter struct, the structure members can refer to Table 3-731. 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_DIRECTTTI;

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 3-779. 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>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel0(TIMERx(x=0..4, 7..13))
<i>TIMER_CH_1</i>	TIMER channel1(TIMERx(x=0..4, 7, 8, 11))
<i>TIMER_CH_2</i>	TIMER channel2(TIMERx(x=0..4, 7))
<i>TIMER_CH_3</i>	TIMER channel3(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,
TIMER_IC_PSC_DIV2);
```

timer_channel_capture_value_register_read

The description of timer_channel_capture_value_register_read is shown as below:

Table 3-780. 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>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	

channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel0(TIMERx(x=0..4, 7..13))
<i>TIMER_CH_1</i>	TIMER channel1(TIMERx(x=0..4, 7, 8, 11))
<i>TIMER_CH_2</i>	TIMER channel2(TIMERx(x=0..4, 7))
<i>TIMER_CH_3</i>	TIMER channel3(TIMERx(x=0..4, 7))
Output parameter{out}	
-	-
Return value	
uint32_t	channel capture compare register value, 0~65535

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 3-781. 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 channel0
<i>TIMER_CH_1</i>	TIMER channel1
Input parameter{in}	
icpwm	TIMER channel input parameter struct, the structure members can refer to Table 3-731. 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_DIRECTTTI;

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 3-782. 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
TIMERx(x=0..4, 7)	TIMER peripheral selection
Input parameter{in}	
hallmode	TIMER hall sensor mode state
TIMER_HALLINTERFACE_ENABLE	TIMER hall sensor mode enable
TIMER_HALLINTERFACE_DISABLE	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 3-783. 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
<i>TIMERx</i>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
intrigger	trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI0</i>	internal trigger input 0(ITI0), TIMERx(x=0..4, 7, 8, 11)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI1</i>	internal trigger input 0(ITI1), TIMERx(x=0..4, 7, 8, 11)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI2</i>	internal trigger input 0(ITI2), TIMERx(x=0..4, 7, 8, 11)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI3</i>	internal trigger input 0(ITI3), TIMERx(x=0..4, 7, 8, 11)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CIO edge flag(CIOF_ED), TIMERx(x=0..4, 7, 8, 11)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output(CIOFE0), 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_ITI0);
```

timer_master_output_trigger_source_select

The description of timer_master_output_trigger_source_select is shown as below:

Table 3-784. 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 outtrigger);
Function descriptions	select TIMER master mode output trigger source

Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
outrigger	master mode control
<i>TIMER_TRI_OUT_SRC_RESET</i>	reset, when the UPG bit in the <i>TIMERx_SWEVG</i> 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>TIMERx</i> (x=0..7))
<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>TIMERx</i> (x=0..7))
<i>TIMER_TRI_OUT_SRC_UPDATE</i>	update, in this mode the master mode controller selects the update event as TRGO.(<i>TIMERx</i> (x=0..7))
<i>TIMER_TRI_OUT_SRC_CH0</i>	a capture or a compare match occurred in channel 0 as trigger output TRGO.(<i>TIMERx</i> (x=0..4,7))
<i>TIMER_TRI_OUT_SRC_O0CPRE</i>	compare, in this mode the master mode controller selects the O0CPRE signal is used as TRGO(<i>TIMERx</i> (x=0..4,7))
<i>TIMER_TRI_OUT_SRC_O1CPRE</i>	compare, in this mode the master mode controller selects the O1CPRE signal is used as TRGO(<i>TIMERx</i> (x=0..4,7))
<i>TIMER_TRI_OUT_SRC_O2CPRE</i>	compare, in this mode the master mode controller selects the O2CPRE signal is used as TRGO(<i>TIMERx</i> (x=0..4,7))
<i>TIMER_TRI_OUT_SRC_O3CPRE</i>	compare, in this mode the master mode controller selects the O3CPRE signal is used as TRGO(<i>TIMERx</i> (x=0..4,7))
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 3-785. 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
TIMERx(x=0..4, 7, 8, 11)	TIMER peripheral selection
Input parameter{in}	
slavemode	slave mode
TIMER_SLAVE_MODE_DISABLE	slave mode disable
TIMER_QUAD_DECODER_MODE0	quadrature decoder mode 0
TIMER_QUAD_DECODER_MODE1	quadrature decoder mode 1
TIMER_QUAD_DECODER_MODE2	quadrature decoder mode 2
TIMER_SLAVE_MODE_RESTART	restart mode
TIMER_SLAVE_MODE_PAUSE	pause mode
TIMER_SLAVE_MODE_EVENT	event mode
TIMER_SLAVE_MODE_EXTERNAL0	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 3-786. 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, uint32_t

	masterslave);
Function descriptions	configure TIMER master 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}	
masterslave	master slave mode state
<i>TIMER_MASTER_SLAVE_MODE_ENABLE</i>	master slave mode enable
<i>TIMER_MASTER_SLAVE_MODE_DISABLE</i>	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 3-787. 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, uint8_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>DIV4</i>	
<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 3-788. 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, 8, 11)</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 3-789. 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 3-790. 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
TIMEx(x=0..4, 7, 8, 11)	TIMER peripheral selection
Input parameter{in}	
intrigger	trigger selection
TIMER_SMCFG_TRGS EL_ITI0	internal trigger input 0(ITI0)
TIMER_SMCFG_TRGS EL_ITI1	internal trigger input 0(ITI1)
TIMER_SMCFG_TRGS EL_ITI2	internal trigger input 0(ITI2)
TIMER_SMCFG_TRGS EL_ITI3	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(TIMER0, 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 3-791. 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 extrigger, uint16_t expolarity, uint8_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
<i>TIMERx(x=0..4, 7, 8, 11)</i>	TIMER peripheral selection
Input parameter{in}	
extrigger	external trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CI0 edge flag(CIOF_ED)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output(CIOFE0)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI1FE1</i>	channel 1 input Filtered output(CI1FE1)
Input parameter{in}	
expolarity	external trigger polarity
<i>TIMER_IC_POLARITY_</i> <i>RISING</i>	active high or rising edge active
<i>TIMER_IC_POLARITY_</i> <i>FALLING</i>	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 CIOFE0 as external clock input */
```

```
timer_external_trigger_as_external_clock_config(TIMER0,  
TIMER_SMCFG_TRGSEL_CIOFE0, TIMER_IC_POLARITY_RISING, 0);
```

timer_external_clock_mode0_config

The description of timer_external_clock_mode0_config is shown as below:

Table 3-792. 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, uint8_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 3-793. 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, uint8_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_</i>	no divided

<i>OFF</i>	
<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,  
TIMER_ETP_FALLING, 0);
```

timer_external_clock_mode1_disable

The description of timer_external_clock_mode1_disable is shown as below:

Table 3-794. 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(TIMERO);

timer_flag_get

The description of timer_flag_get is shown as below:

Table 3-795. 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>	TIMER peripheral selection (x 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 TIMERO update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get(TIMERO, TIMER_FLAG_UP);
```

timer_flag_clear

The description of timer_flag_clear is shown as below:

Table 3-796. 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>	TIMER peripheral selection (x 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 3-797. 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>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
interrupt	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 3-798. 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>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
interrupt	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, <i>TIMERx</i> (<i>x</i> =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 3-799. 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
<i>TIMERx</i>	TIMER peripheral selection (x please refer to the following parameters)
Input parameter{in}	
interrupt	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, <i>TIMERx</i> (<i>x</i> =0..13)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> (<i>x</i> =0..4, 7..13)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> (<i>x</i> =0..4, 7, 8, 11)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_INT_FLAG_CMT</i>	channel commutation interrupt flag, <i>TIMERx</i> (<i>x</i> =0, 7)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> (<i>x</i> =0, 7, 8, 11)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, <i>TIMERx</i> (<i>x</i> =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 3-800. Function timer_interrupt_flag_clear

Function name	timer_interrupt_flag_clear
Function prototype	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt);
Function descriptions	clear TIMER interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection (x 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_CMT</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);
```

3.26. TLI

The TFT(LCD) display interface provides a parallel digital RGB (Red, Green, Blue) and signals for horizontal, vertical synchronization, Pixel Clock and Data Enable as output to interface directly to a variety of LCD(Liquid Crystal Display) and TFT(Thin Film Transistor) panels. The TLI registers are listed in chapter [3.26.1](#), the TLI firmware functions are introduced in chapter [3.26.2](#).

3.26.1. Descriptions of Peripheral registers

TLI registers are listed in the table shown as below:

Table 3-801. 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_CPPOS	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_LxFLEN	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

3.26.2. Descriptions of Peripheral functions

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

Table 3-802. 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 display timing parameters

Function name	Function description
tli_dither_config	configure TLI dither function
tli_enable	enable TLI
tli_disable	disable TLI
tli_reload_config	configure 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	TLI layer initialize
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	TLI layer LUT initialize
tli_color_key_init	initialize TLI layer color key
tli_layer_enable	TLI layer enable
tli_layer_disable	TLI layer disable
tli_color_key_enable	TLI layer color keying enable
tli_color_key_disable	TLI layer color keying disable
tli_lut_enable	TLI layer LUT enable
tli_lut_disable	TLI layer LUT disable
tli_line_mark_set	set line mark value
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

Table 3-803. 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_htsiz	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

Table 3-804. 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	packeted 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

Table 3-805. 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

Enum tli_layer_ppf_enum

Table 3-806. Enum tli_layer_ppf_enum

Member name	Function description
LAYER_PPF_ARGB8888	layerx packeted pixel format ARGB8888

LAYER_PPF_RGB888	layerx packeted pixel format RGB888
LAYER_PPF_RGB565	layerx packeted pixel format RGB565
LAYER_PPF_ARGB1555	layerx packeted pixel format ARGB1555
LAYER_PPF_ARGB4444	layerx packeted pixel format ARGB4444
LAYER_PPF_L8	layerx packeted pixel format L8
LAYER_PPF_AL44	layerx packeted pixel format AL44
LAYER_PPF_AL88	layerx packeted pixel format AL88

tli_deinit

The description of tli_deinit is shown as below:

Table 3-807. Function tli_deinit

Function name	tli_deinit
Function prototype	void tli_deinit (void);
Function descriptions	deinitialize TLI registers
Precondition	-
The called functions	rcu_bkp_reset_enable / rcu_bkp_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize TLI registers */
tli_deinit();
```

tli_struct_para_init

The description of tli_struct_para_init is shown as below:

Table 3-808. 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}	
-	-

Output parameter{out}	
*tli_struct	a pointer to tli_parameter_struct
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 3-809. 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	the data needed to initialize TLI
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TLI display timing parameters */
```

```
tli_parameter_struct tli_init_struct;
```

```
/* TLI initialization */
```

```
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_pixelclk = 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 = 42 + 480;

tli_init_struct.activesz_vasz = 11 + 272;

tli_init_struct.totalsz_htsiz = 42 + 480 + 2;

tli_init_struct.totalsz_vtsiz = 11 + 272 + 2;

/* LCD background color configure */

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 3-810. Function tli_dither_config

Function name	tli_dither_config
Function prototype	void tli_dither_config(uint8_t ditherstat);
Function descriptions	configure TLI dither function
Precondition	-
The called functions	-
Input parameter{in}	
ditherstat	Dither status
TLI_DITHER_ENABLE	TLI dither enable
TLI_DITHER_DISABLE	TLI dither disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TLI dither function */

tli_dither_config(TLI_DITHER_ENABLE);

```

tli_enable

The description of tli_enable is shown as below:

Table 3-811. Function tli_enable

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

Example:

```
/* enable TLI */
```

```
tli_enable( );
```

tli_disable

The description of tli_disable is shown as below:

Table 3-812. Function tli_disable

Function name	tli_disable
Function prototype	void tli_disable(void);
Function descriptions	TLI disable
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 3-813. Function tli_reload_config

Function name	tli_reload_config
---------------	-------------------

Function prototype	void tli_reload_config(uint8_t reloadmode);
Function descriptions	configure TLI reload mode
Precondition	-
The called functions	-
Input parameter{in}	
reloadmode	Reload mode
<i>TLI_FRAME_BLANK_RELOAD_EN</i>	the layer configuration will be reloaded at frame blank
<i>TLI_REQUEST_RELOAD_EN</i>	the layer configuration will be reloaded after this bit sets
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TLI reload mode */
```

```
tli_reload_config (TLI_FRAME_BLANK_RELOAD_EN);
```

tli_layer_struct_para_init

The description of tli_layer_struct_para_init is shown as below:

Table 3-814. 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}	
-	-
Output parameter{out}	
*layer_struct	a pointer to tli_layer_parameter_struct
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 3-815. 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	TLI layer initialize
Precondition	-
The called functions	-
Input parameter{in}	
layerx	Layer base address
LAYER0	Layer0 base address
LAYER1	Layer1 base address
Input parameter{in}	
layer_struct	TLI Layer parameter struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* TLI layer initialize */

tli_layer_parameter_struct  tli_layer_init_struct;

/* tli layer1 configuration */

/* tli window size configuration */

tli_layer_init_struct.layer_window_leftpos = 20 + 43;
tli_layer_init_struct.layer_window_rightpos = (20 + 140 + 43 - 1);
tli_layer_init_struct.layer_window_toppos = 30 + 12;
tli_layer_init_struct.layer_window_bottompos = (30 + 60 + 12 - 1);

/* tli window pixel format configuration */

tli_layer_init_struct.layer_ppf = LAYER_PPF_RGB565;

/* tli window specified alpha configuration */

tli_layer_init_struct.layer_sa = 255;

/* tli window blend configuration */

tli_layer_init_struct.layer_acf1 = LAYER_ACF1_PASA;
tli_layer_init_struct.layer_acf2 = LAYER_ACF2_PASA;

```

```

/* tli layer default alpha R,G,B value configuration */

tli_layer_init_struct.layer_default_alpha = 0;

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 frame buffer base address configuration */

tli_layer_init_struct.layer_frame_bufaddr = (uint32_t)&image_0;

tli_layer_init_struct.layer_frame_line_length = ((480 * 2) + 3);

tli_layer_init_struct.layer_frame_buf_stride_offset = (480 * 2);

tli_layer_init_struct.layer_frame_total_line_number = 60;

tli_layer_init(LAYER1, &tli_layer_init_struct);

```

tli_layer_window_offset_modify

The description of tli_layer_window_offset_modify is shown as below:

Table 3-816. Function tli_layer_window_offset_modify

Function name	tli_layer_window_offset_modify
Function prototype	void tli_layer_window_offset_modify(uint32_t layerx,uint32_t offset_x,uint32_t offset_y)
Function descriptions	reconfigure window position
Precondition	-
The called functions	-
Input parameter{in}	
layerx	Layer base address
LAYER0	Layer0 base address
LAYER1	Layer1 base address
Input parameter{in}	
offset_x	new horizontal offset
Input parameter{in}	
offset_y	new vertical offset
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* TLI layer initialize */

tli_layer_window_offset_modify(LAYER0, 0x40 , 0x40);

```

tli_lut_struct_para_init

The description of tli_lut_struct_para_init is shown as below:

Table 3-817. 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}	
-	-
Output parameter{out}	
*lut_struct	a pointer to tli_layer_lut_parameter_struct
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 3-818. 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	TLI layer LUT initialize
Precondition	-
The called functions	-
Input parameter{in}	
layerx	Layer base address
<i>LAYER0</i>	Layer0 base address
<i>LAYER1</i>	Layer1 base address
Input parameter{in}	
lut_struct	TLI layer LUT parameter struct
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* TLI layer initialize */

tli_layer_lut_parameter_struct lut_struct;

lut_struct.layer_table_addr = 0x20000400;

lut_struct.layer_lut_channel_red = 0x20;

lut_struct.layer_lut_channel_green = 0x40;

lut_struct.layer_lut_channel_blue = 0x40;

tli_layer_init(LAYER0, &lut_struct);
```

tli_color_key_init

The description of tli_color_key_init is shown as below:

Table 3-819. Function tli_ckey_init

Function name	tli_ckey_init
Function prototype	void tli_color_key_init(uint32_t layerx,uint32_t redkey,uint32_t greenkey,uint32_t bluekey)
Function descriptions	TLI layer color key initialize
Precondition	-
The called functions	-
Input parameter{in}	
layerx	Layer base address
LAYER0	Layer0 base address
LAYER1	Layer1 base address
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:

```
/* TLI layer color key initialize */

tli_color_key_init (LAYER0, 0xAA, 0xFF, 0x00);
```

tli_layer_enable

The description of tli_layer_enable is shown as below:

Table 3-820. Function tli_layer_enable

Function name	tli_layer_enable
Function prototype	void tli_layer_enable(uint32_t layerx);
Function descriptions	TLI layer enable
Precondition	-
The called functions	-
Input parameter{in}	
layerx	Layer base address
<i>LAYER0</i>	Layer0 base address
<i>LAYER1</i>	Layer1 base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* TLI layer enable */
```

```
tli_layer_enable(LAYER0);
```

tli_layer_disable

The description of tli_layer_disable is shown as below:

Table 3-821. Function tli_layer_disable

Function name	tli_layer_disable
Function prototype	void tli_layer_disable(uint32_t layerx);
Function descriptions	TLI layer disable
Precondition	-
The called functions	-
Input parameter{in}	
layerx	Layer base address
<i>LAYER0</i>	Layer0 base address
<i>LAYER1</i>	Layer1 base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* TLI layer disable */
```



```
tli_layer_disable(LAYER0);
```

tli_color_key_enable

The description of tli_color_key_enable is shown as below:

Table 3-822. Function tli_color_key_enable

Function name	tli_color_key_enable
Function prototype	void tli_color_key_enable(uint32_t layerx);
Function descriptions	TLI layer color keying enable
Precondition	-
The called functions	-
Input parameter{in}	
layerx	Layer base address
LAYER0	Layer0 base address
LAYER1	Layer1 base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* TLI layer color keying enable */
tli_color_key_enable(LAYER0);
```

tli_color_key_disable

The description of tli_color_key_disable is shown as below:

Table 3-823. Function tli_color_key_disable

Function name	tli_color_key_disable
Function prototype	void tli_color_key_disable(uint32_t layerx);
Function descriptions	TLI layer color keying disable
Precondition	-
The called functions	-
Input parameter{in}	
layerx	Layer base address
LAYER0	Layer0 base address
LAYER1	Layer1 base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* TLI layer color keying disable */
```

```
tli_color_key_disable(LAYER0);
```

tli_lut_enable

The description of tli_lut_enable is shown as below:

Table 3-824. Function tli_lut_enable

Function name	tli_lut_enable
Function prototype	void tli_lut_enable(uint32_t layerx);
Function descriptions	TLI layer LUT enable
Precondition	-
The called functions	-
Input parameter{in}	
layerx	Layer base address
LAYER0	Layer0 base address
LAYER1	Layer1 base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* TLI layer LUT enable */
```

```
tli_lut_enable(LAYER0);
```

tli_lut_disable

The description of tli_lut_disable is shown as below:

Table 3-825. Function tli_lut_disable

Function name	tli_lut_disable
Function prototype	void tli_lut_disable(uint32_t layerx);
Function descriptions	TLI layer LUT disable
Precondition	-
The called functions	-
Input parameter{in}	
layerx	Layer base address
LAYER0	Layer0 base address
LAYER1	Layer1 base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* TLI layer LUT disable */

tli_lut_disable(LAYER0);
```

tli_line_mark_set

The description of tli_line_mark_set is shown as below:

Table 3-826. 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( 0x40);
```

tli_current_pos_get

The description of tli_current_pos_get is shown as below:

Table 3-827. 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	0x0 – 0xFFFFFFFF

Example:

```
/* get current displayed position */
```

```
uint32_t pos = tli_current_pos_get( );
```

tli_interrupt_enable

The description of tli_interrupt_enable is shown as below:

Table 3-828. 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
TLI_INT_LM	line mark interrupt
TLI_INT_FE	FIFO error interrupt
TLI_INT_TE:	transaction error interrupt
TLI_INT_LCR	layer configuration reloaded interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TLI interrupt */
```

```
tli_interrupt_enable (TLI_INT_LM);
```

tli_interrupt_disable

The description of tli_interrupt_disable is shown as below:

Table 3-829. 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
TLI_INT_LM	line mark interrupt
TLI_INT_FE	FIFO error interrupt
TLI_INT_TE:	transaction error interrupt
TLI_INT_LCR	layer configuration reloaded interrupt

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TLI interrupt */
```

```
tli_interrupt_disable (TLI_INT_LM);
```

tli_interrupt_flag_get

The description of tli_interrupt_flag_get is shown as below:

Table 3-830. 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
TLI_INT_FLAG_LM	line mark interrupt flag
TLI_INT_FLAG_FE	FIFO error interrupt flag
TLI_INT_FLAG_TE	transaction error interrupt flag
TLI_INT_FLAG_LCR:	layer configuration reloaded interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get TLI flag state */
```

```
FlagStatus status;
```

```
status = tli_interrupt_flag_get (TLI_INT_FLAG_LM);
```

tli_interrupt_flag_clear

The description of tli_interrupt_flag_clear is shown as below:

Table 3-831. 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}	
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	
-	-

Example:

```
/* clear TLI flag state */
```

```
tli_interrupt_flag_clear (TLI_INT_FLAG_LM);
```

tli_flag_get

The description of tli_flag_get is shown as below:

Table 3-832. 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
<i>TLI_FLAG_VDE</i>	current VDE state
<i>TLI_FLAG_HDE</i>	current HDE state
<i>TLI_FLAG_VS</i>	current vs state
<i>TLI_FLAG_HS</i>	current hs state
<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:

```
/* get TLI flag state */
```

```
FlagStatus status;
```

```
status = tli_flag_get (TLI_FLAG_VDE);
```

3.27. TRNG

The true random number generator (TRNG) module can generate a 32-bit value using continuous analog noise. The TRNG registers are listed in chapter [3.27.1](#). the TRNG firmware functions are introduced in chapter [3.27.2](#).

3.27.1. Descriptions of Peripheral registers

TRNG registers are listed in the table shown as below:

Table 3-833. TRNG Registers

Registers	Descriptions
TRNG_CTL	TRNG control register
TRNG_STAT	TRNG status register
TRNG_DATA	TRNG data register

3.27.2. Descriptions of Peripheral functions

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

Table 3-834. 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_flag_get	get the trng status flags
trng_interrupt_enable	the trng interrupt enable
trng_interrupt_disable	the trng interrupt disable
trng_interrupt_flag_get	get the trng interrupt flags
trng_interrupt_flag_clear	clear the trng interrupt flags

Enum trng_flag_enum

Table 3-835. Enum trng_flag_enum

Member name	Function description
TRNG_FLAG_DRDY	random data ready status
TRNG_FLAG_CECS	clock error current status
TRNG_FLAG_SECS	seed error current status

Enum trng_int_flag_enum

Table 3-836. Enum trng_int_flag_enum

Member name	Function description
TRNG_INT_FLAG_CE	clock error interrupt flag
TRNG_INT_FLAG_SE	seed error interrupt flag

trng_deinit

The description of trng_deinit is shown as below:

Table 3-837. Function trng_deinit

Function name	trng_deinit
Function prototype	void trng_deinit (void);
Function descriptions	TRNG deinit
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* TRNG deinit */
```

```
trng_deinit();
```

trng_enable

The description of trng_enable is shown as below:

Table 3-838. Function trng_enable

Function name	trng_enable
Function prototype	void trng_enable(void);
Function descriptions	enable the TRNG
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TRNG interface */
```

```
trng_enable();
```

trng_disable

The description of trng_disable is shown as below:

Table 3-839. Function trng_disable

Function name	trng_disable
Function prototype	void trng_disable(void);
Function descriptions	disable the TRNG
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TRNG interface */
```

```
trng_disable();
```

trng_get_true_random_data

The description of trng_get_true_random_data is shown as below:

Table 3-840. 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	0x0 – 0xFFFFFFFF

Example:

```
/* get the true random data */
```

```
uint32_t data = trng_get_true_random_data();
```

trng_flag_get

The description of trng_flag_get is shown as below:

Table 3-841. 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, refer to Table 3-835. Enum trng_flag_enum
TRNG_FLAG_DRDY	random Data ready status
TRNG_FLAG_CECS	clock error current status
TRNG_FLAG_SECS	seed error current status
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the trng status flags */
```

```
FlagStatus trng_flag = RESET;
```

```
trng_flag = trng_flag_get(TRNG_FLAG_DRDY);
```

trng_interrupt_enable

The description of trng_interrupt_enable is shown as below:

Table 3-842. Function trng_interrupt_enable

Function name	trng_interrupt_enable
Function prototype	void trng_interrupt_enable(void);
Function descriptions	enable the TRNG interrupt
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable the TRNG interrupt */
```

```
trng_interrupt_enable();
```

trng_interrupt_disable

The description of trng_interrupt_disable is shown as below:

Table 3-843. Function trng_interrupt_disable

Function name	trng_interrupt_disable
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 the TRNG interrupt */
```

```
trng_interrupt_disable();
```

trng_interrupt_flag_get

The description of trng_interrupt_flag_get is shown as below:

Table 3-844. 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 flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	trng interrupt flag, refer to Table 3-836. Enum trng_int_flag_enum
TRNG_INT_FLAG_CE	clock error interrupt flag
TRNG_INT_FLAG_SE	seed error interrupt flag
Output parameter{out}	
-	-
Return value	

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

Example:

```
/* get the trng interrupt flag */

FlagStatus trng_flag = RESET;

trng_flag = trng_interrupt_flag_get(TRNG_INT_FLAG_CE);
```

trng_interrupt_flag_clear

The description of trng_interrupt_flag_clear is shown as below:

Table 3-845. Function trng_interrupt_flag_clear

Function name	trng_interrupt_flag_clear
Function prototype	void trng_interrupt_flag_clear(trng_int_flag_enum int_flag);
Function descriptions	clear the trng interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	trng interrupt flag, refer to Table 3-836. Enum trng_int_flag_enum
TRNG_INT_FLAG_CE	clock error interrupt flag
TRNG_INT_FLAG_SE	seed error interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*clear the trng interrupt flag */

trng_interrupt_flag_clear(TRNG_INT_FLAG_CE);
```

3.28. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.28.1](#), the USART firmware functions are introduced in chapter [3.28.2](#).

3.28.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

Table 3-846. 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

3.28.2. Descriptions of Peripheral functions

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

Table 3-847. USART firmware function

Function name	Function description
usart_deinit	reset USART/UART
usart_baudrate_set	configure USART baud rate value
usart_parity_config	configure USART parity
usart_word_length_set	configure USART word length
usart_stop_bit_set	configure USART stop bit length
usart_enable	enable USART
usart_disable	disable USART
usart_transmit_config	configure USART transmitter
usart_receive_config	configure USART receiver
usart_data_first_config	data is transmitted/received with the LSB/MSB first
usart_invert_config	configure USART inversion
usart_receiver_timeout_enable	enable receiver timeout
usart_receiver_timeout_disable	disable receiver timeout
usart_receiver_timeout_threshold_config	configure receiver timeout threshold
usart_data_transmit	USART transmit data function
usart_data_receive	USART receive data function
usart_address_config	configure the address of the USART in wake up by address match mode
usart_mute_mode_enable	enable mute mode
usart_mute_mode_disable	disable mute mode
usart_mute_mode_wakeup_config	configure wakeup method in mute mode
usart_lin_mode_enable	enable LIN mode
usart_lin_mode_disable	disable LIN mode

Function name	Function description
usart_lin_break_dection_length_config	configure lin break frame length
usart_send_break	send break frame
usart_halfduplex_enable	enable half-duplex mode
usart_halfduplex_disable	disable half-duplex mode
usart_synchronous_clock_enable	enable CK pin in synchronous mode
usart_synchronous_clock_disable	disable CK pin in synchronous mode
usart_synchronous_clock_config	configure USART synchronous mode parameters
usart_guard_time_config	configure guard time value in smartcard mode
usart_smartcard_mode_enable	enable smartcard mode
usart_smartcard_mode_disable	disable smartcard mode
usart_smartcard_mode_nack_enable	enable NACK in smartcard mode
usart_smartcard_mode_nack_disable	disable NACK in smartcard mode
usart_smartcard_autoretry_config	configure smartcard auto-retry number
usart_block_length_config	configure block length
usart_irda_mode_enable	enable IrDA mode
usart_irda_mode_disable	disable IrDA mode
usart_prescaler_config	configure the peripheral clock prescaler
usart_irda_lowpower_config	configure IrDA low-power
usart_hardware_flow_rts_config	configure hardware flow control RTS
usart_hardware_flow_cts_config	configure hardware flow control CTS
usart_dma_receive_config	configure USART DMA reception
usart_dma_transmit_config	configure USART DMA transmission
usart_flag_get	get flag in STAT0/STAT1 register
usart_flag_clear	clear flag in STAT0/STAT1 register
usart_interrupt_enable	enable USART interrupt
usart_interrupt_disable	disable USART interrupt
usart_interrupt_flag_get	get USART interrupt and flag status
usart_interrupt_flag_clear	clear USART interrupt flag in STAT0/STAT1 register

Enum usart_flag_enum

Table 3-848. Enum usart_flag_enum

Member name	Function description
USART_FLAG_CTS	CTS change flag
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_TBE	transmit data buffer empty
USART_FLAG_TC	transmission complete
USART_FLAG_RBNE	read data buffer not empty
USART_FLAG_IDLE	IDLE line detected flag
USART_FLAG_ORERR	overrun error flag
USART_FLAG_NERR	noise error flag

Member name	Function description
USART_FLAG_FERR	frame error flag
USART_FLAG_PERR	parity error flag
USART_FLAG_BSY	busy flag
USART_FLAG_EB	end of block flag
USART_FLAG_RT	receiver timeout flag

Enum usart_interrupt_flag_enum

Table 3-849. Enum usart_interrupt_flag_enum

Member name	Function description
USART_INT_FLAG_PERR	parity error interrupt flag
USART_INT_FLAG_TBE	transmitter buffer empty interrupt flag
USART_INT_FLAG_TC	transmission complete interrupt flag
USART_INT_FLAG_RBNE	read data buffer not empty interrupt flag
USART_INT_FLAG_RBNE_ORE RR	read data buffer not empty interrupt and overrun error flag
USART_INT_FLAG_IDLE	IDLE line detected interrupt flag
USART_INT_FLAG_LBD	LIN break detected interrupt flag
USART_INT_FLAG_CTS	CTS interrupt flag
USART_INT_FLAG_ERR_ORER R	overrun error interrupt flag
USART_INT_FLAG_ERR_NERR	noise error interrupt flag
USART_INT_FLAG_ERR_FERR	frame error interrupt flag
USART_INT_FLAG_EB	end of block interrupt flag
USART_INT_FLAG_RT	receive timeout interrupt flag

Enum usart_interrupt_enum

Table 3-850. Enum usart_interrupt_enum

Member name	Function description
USART_INT_PERR	parity error interrupt
USART_INT_TBE	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_CTS	CTS interrupt
USART_INT_ERR	error interrupt
USART_INT_EB	end of block interrupt
USART_INT_RT	receive timeout interrupt

Enum usart_invert_enum

Table 3-851. Enum usart_invert_enum

Member name	Function description
USART_DINV_ENABLE	data bit level inversion
USART_DINV_DISABLE	data bit level not inversion
USART_TXPIN_ENABLE	TX pin level inversion
USART_TXPIN_DISABLE	TX pin level not inversion
USART_RXPIN_ENABLE	RX pin level inversion
USART_RXPIN_DISABLE	RX pin level not inversion

usart_deinit

The description of usart_deinit is shown as below:

Table 3-852. 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	usart peripheral
USARTx	x=0,1,2,5
UARTx	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 3-853. 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	usart 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 */
usart_baudrate_set(USART0, 115200);
```

usart_parity_config

The description of usart_parity_config is shown as below:

Table 3-854. Function usart_parity_config

Function name	usart_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	usart peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
paritycfg	USART/UART parity
<i>USART_PM_NONE</i>	no parity
<i>USART_PM_ODD</i>	odd parity
<i>USART_PM_EVEN</i>	even parity
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 parity */
usart_parity_config(USART0, USART_PM_EVEN);
```

usart_word_length_set

The description of usart_word_length_set is shown as below:

Table 3-855. Function usart_word_length_set

Function name	usart_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	usart peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
wlen	USART word length
USART_WL_8BIT	8 bits
USART_WL_9BIT	9 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 word length */
usart_word_length_set(USART0, USART_WL_9BIT);
```

usart_stop_bit_set

The description of usart_stop_bit_set is shown as below:

Table 3-856. Function usart_stop_bit_set

Function name	usart_stop_bit_set
Function prototype	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
Function descriptions	configure USART stop bit length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
stblen	USART stop bit
USART_STB_1BIT	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 UARTx(x=3,4,6,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 stop bit length */
usart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

usart_enable

The description of usart_enable is shown as below:

Table 3-857. Function usart_enable

Function name	usart_enable
Function prototype	void usart_enable(uint32_t usart_periph);
Function descriptions	enable USART
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart 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 */
usart_enable(USART0);
```

usart_disable

The description of usart_disable is shown as below:

Table 3-858. Function usart_disable

Function name	usart_disable
Function prototype	void usart_disable(uint32_t usart_periph);
Function descriptions	disable USART
Precondition	-

The called functions	-
Input parameter{in}	
usart_periph	usart 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 */
```

```
usart_disable(USART0);
```

usart_transmit_config

The description of usart_transmit_config is shown as below:

Table 3-859. Function usart_transmit_config

Function name	usart_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	usart peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
txconfig	enable or disable USART transmitter
<i>USART_TRANSMIT_ENABLE</i>	enable USART transmission
<i>USART_TRANSMIT_DISABLE</i>	disable USART transmission
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 transmitter */
```

```
usart_transmit_config(USART0, USART_TRANSMIT_ENABLE);
```

usart_receive_config

The description of usart_receive_config is shown as below:

Table 3-860. Function usart_receive_config

Function name	usart_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	usart 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 */
```

```
usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

usart_data_first_config

The description of usart_data_first_config is shown as below:

Table 3-861. Function usart_data_first_config

Function name	usart_data_first_config
Function prototype	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
Function descriptions	data is transmitted/received with the LSB/MSB first
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2,5
Input parameter{in}	
msbf	LSB/MSB

<i>USART_MSBF_LSB</i>	LSB first
<i>USART_MSBF_MSB</i>	MSB first
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* data is transmitted/received with the LSB first */
```

```
usart_data_first_config(USART0, USART_MSBF_LSB);
```

usart_invert_config

The description of usart_invert_config is shown as below:

Table 3-862. Function usart_invert_config

Function name	usart_invert_config
Function prototype	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
Function descriptions	configure USART inversion
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
invertpara	inversion parameters, refer to Table 3-851. Enum usart_invert_enum .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 inversion */
```

```
usart_invert_config(USART0, USART_DINV_ENABLE);
```

usart_receiver_timeout_enable

The description of usart_receiver_timeout_enable is shown as below:

Table 3-863. Function usart_receiver_timeout_enable

Function name	usart_receiver_timeout_enable
Function prototype	void usart_receiver_timeout_enable(uint32_t usart_periph);
Function descriptions	enable receiver timeout

Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 receiver timeout */
usart_receiver_timeout_enable(USART0);
```

usart_receiver_timeout_disable

The description of usart_receiver_timeout_disable is shown as below:

Table 3-864. Function usart_receiver_timeout_disable

Function name	usart_receiver_timeout_disable
Function prototype	void usart_receiver_timeout_disable(uint32_t usart_periph);
Function descriptions	disable receiver timeout
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 receiver timeout */
usart_receiver_timeout_disable(USART0);
```

usart_receiver_timeout_threshold_config

The description of usart_receiver_timeout_threshold_config is shown as below:

Table 3-865. Function usart_receiver_timeout_threshold_config

Function name	usart_receiver_timeout_threshold_config
Function prototype	void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);

Function descriptions	configure the receiver timeout threshold
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
rtimeout	0-0xFFFFF
Output parameter{out}	
Return value	
-	-

Example:

```
/* set the receiver timeout threshold of USART0 */
```

```
usart_receiver_timeout_threshold_config(USART0, 0xFFFF);
```

usart_data_transmit

The description of usart_data_transmit is shown as below:

Table 3-866. Function usart_data_transmit

Function name	usart_data_transmit
Function prototype	void usart_data_transmit(uint32_t usart_periph, uint16_t data);
Function descriptions	USART transmit data function
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
data	data of transmission
<i>0-0xFF</i>	data of transmission
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 transmit data */
```

```
usart_data_transmit(USART0, 0xAA);
```


usart_data_receive

The description of usart_data_receive is shown as below:

Table 3-867. Function usart_data_receive

Function name	usart_data_receive
Function prototype	uint16_t usart_data_receive(uint32_t usart_periph);
Function descriptions	USART receive data function
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
uint16_t	data of received (0-0xFF)

Example:

```
/* USART0 receive data */
```

```
uint16_t temp;
```

```
temp = usart_data_receive(USART0);
```

usart_address_config

The description of usart_address_config is shown as below:

Table 3-868. Function usart_address_config

Function name	usart_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	usart 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 3-869. 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	usart peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 receiver mute mode */
```

```
usart_mute_mode_enable(USART0);
```

usart_mute_mode_disable

The description of usart_mute_mode_disable is shown as below:

Table 3-870. Function usart_mute_mode_disable

Function name	usart_mute_mode_disable
Function prototype	void usart_mute_mode_disable(uint32_t usart_periph);
Function descriptions	disable mute mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 receiver mute mode */
```

```
usart_mute_mode_disable(USART0);
```

usart_mute_mode_wakeup_config

The description of usart_mute_mode_wakeup_config is shown as below:

Table 3-871. Function usart_mute_mode_wakeup_config

Function name	usart_mute_mode_wakeup_config
Function prototype	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
Function descriptions	configure wakeup method in mute mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
wmethod	two methods be used to enter or exit the mute mode
USART_WM_IDLE	idle line
USART_WM_ADDR	address mask
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 wakeup method in mute mode */
```

```
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

usart_lin_mode_enable

The description of usart_lin_mode_enable is shown as below:

Table 3-872. Function usart_lin_mode_enable

Function name	usart_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	usart 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 LIN mode */
```

```
usart_lin_mode_enable(USART0);
```

usart_lin_mode_disable

The description of usart_lin_mode_disable is shown as below:

Table 3-873. Function usart_lin_mode_disable

Function name	usart_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	usart 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 LIN mode */
```

```
usart_lin_mode_disable(USART0);
```

usart_lin_break_dection_length_config

The description of usart_lin_break_dection_length_config is shown as below:

Table 3-874. Function `usart_lin_break_dection_length_config`

Function name	<code>usart_lin_break_dection_length_config</code>
Function prototype	<code>void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lblen);</code>
Function descriptions	configure lin break frame length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
lblen	lin break frame length
<i>USART_LBLEN_10B</i>	10 bits
<i>USART_LBLEN_11B</i>	11 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure LIN break frame length */
```

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

usart_send_break

The description of `usart_send_break` is shown as below:

Table 3-875. Function `usart_send_break`

Function name	<code>usart_send_break</code>
Function prototype	<code>void usart_send_break(uint32_t usart_periph);</code>
Function descriptions	send break frame
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart 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 3-876. 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	usart peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 half duplex mode*/
```

```
usart_halfduplex_enable(USART0);
```

usart_halfduplex_disable

The description of usart_halfduplex_disable is shown as below:

Table 3-877. Function usart_halfduplex_disable

Function name	usart_halfduplex_disable
Function prototype	void usart_halfduplex_disable(uint32_t usart_periph);
Function descriptions	disable half-duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 half duplex mode*/

usart_halfduplex_disable(USART0);
```

usart_synchronous_clock_enable

The description of usart_synchronous_clock_enable is shown as below:

Table 3-878. Function usart_synchronous_clock_enable

Function name	usart_synchronous_clock_enable
Function prototype	void usart_synchronous_clock_enable(uint32_t usart_periph);
Function descriptions	enable CK pin in synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 CK pin in synchronous mode */

usart_synchronous_clock_enable(USART0);
```

usart_synchronous_clock_disable

The description of usart_synchronous_clock_disable is shown as below:

Table 3-879. Function usart_synchronous_clock_disable

Function name	usart_synchronous_clock_disable
Function prototype	void usart_synchronous_clock_disable(uint32_t usart_periph);
Function descriptions	disable CK pin in synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 CK pin in synchronous mode */
```

```
usart_synchronous_clock_disable(USART0);
```

usart_synchronous_clock_config

The description of usart_synchronous_clock_config is shown as below:

Table 3-880. Function usart_synchronous_clock_config

Function name	usart_synchronous_clock_config
Function prototype	void usart_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	usart 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
USART_CPH_1CK	first clock transition is the first data capture edge
USART_CPH_2CK	second clock transition is the first data capture edge
Input parameter{in}	
cpl	clock polarity
USART_CPL_LOW	steady low value on CK pin
USART_CPL_HIGH	steady high value on CK pin
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 synchronous mode parameters */
```

```
usart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,  
USART_CPL_HIGH);
```

usart_guard_time_config

The description of usart_guard_time_config is shown as below:

Table 3-881. Function `usart_guard_time_config`

Function name	<code>usart_guard_time_config</code>
Function prototype	<code>void usart_guard_time_config(uint32_t usart_periph, uint32_t guat);</code>
Function descriptions	configure guard time value in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	usart peripheral
<code>USARTx</code>	x=0,1,2,5
Input parameter{in}	
<code>guat</code>	guard time value
<code>0-0x000000FF</code>	guard time value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 guard time value in smartcard mode */
usart_guard_time_config(USART0, 0x00000055);
```

`usart_smartcard_mode_enable`

The description of `usart_smartcard_mode_enable` is shown as below:

Table 3-882. Function `usart_smartcard_mode_enable`

Function name	<code>usart_smartcard_mode_enable</code>
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}	
<code>usart_periph</code>	usart peripheral
<code>USARTx</code>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 smartcard mode */
usart_smartcard_mode_enable(USART0);
```

usart_smartcard_mode_disable

The description of usart_smartcard_mode_disable is shown as below:

Table 3-883. Function usart_smartcard_mode_disable

Function name	usart_smartcard_mode_disable
Function prototype	void usart_smartcard_mode_disable(uint32_t usart_periph);
Function descriptions	disable smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 smartcard mode */
usart_smartcard_mode_disable(USART0);
```

usart_smartcard_mode_nack_enable

The description of usart_smartcard_mode_nack_enable is shown as below:

Table 3-884. Function usart_smartcard_mode_nack_enable

Function name	usart_smartcard_mode_nack_enable
Function prototype	void usart_smartcard_mode_nack_enable(uint32_t usart_periph);
Function descriptions	enable NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_enable(USART0);
```

usart_smartcard_mode_nack_disable

The description of usart_smartcard_mode_nack_disable is shown as below:

Table 3-885. Function usart_smartcard_mode_nack_disable

Function name	usart_smartcard_mode_nack_disable
Function prototype	void usart_smartcard_mode_nack_disable(uint32_t usart_periph);
Function descriptions	disable NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_disable(USART0);
```

usart_smartcard_autoretry_config

The description of usart_smartcard_autoretry_config is shown as below:

Table 3-886. Function usart_smartcard_autoretry_config

Function name	usart_smartcard_autoretry_config
Function prototype	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);
Function descriptions	configure smartcard auto-retry number
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
scrtnum	smartcard auto-retry number
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 smartcard auto-retry number */
```

```
usart_smartcard_autoretry_config(USART0, 0x00000005);
```

usart_block_length_config

The description of usart_block_length_config is shown as below:

Table 3-887. Function usart_block_length_config

Function name	usart_block_length_config
Function prototype	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
Function descriptions	configure block length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
bl	block length
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 block length in Smartcard T=1 reception */
```

```
usart_block_length_config(USART0, 0x00000005);
```

usart_irda_mode_enable

The description of usart_irda_mode_enable is shown as below:

Table 3-888. Function usart_irda_mode_enable

Function name	usart_irda_mode_enable
Function prototype	void usart_irda_mode_enable(uint32_t usart_periph);
Function descriptions	enable IrDA mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart 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 */
```

```
usart_irda_mode_enable(USART0);
```

usart_irda_mode_disable

The description of usart_irda_mode_disable is shown as below:

Table 3-889. Function usart_irda_mode_disable

Function name	usart_irda_mode_disable
Function prototype	void usart_irda_mode_disable(uint32_t usart_periph);
Function descriptions	disable IrDA mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 IrDA mode */
```

```
usart_irda_mode_disable(USART0);
```

usart_prescaler_config

The description of usart_prescaler_config is shown as below:

Table 3-890. Function usart_prescaler_config

Function name	usart_prescaler_config
Function prototype	void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);
Function descriptions	c configure the peripheral clock prescaler
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	

psc	clock prescaler
<i>0x00-0xFF</i>	clock prescaler
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
```

```
usart_prescaler_config(USART0, 0x00);
```

usart_irda_lowpower_config

The description of usart_irda_lowpower_config is shown as below:

Table 3-891. Function usart_irda_lowpower_config

Function name	usart_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	usart 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_NORMAL</i>	normal
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

usart_hardware_flow_rts_config

The description of usart_hardware_flow_rts_config is shown as below:

Table 3-892. Function `usart_hardware_flow_rts_config`

Function name	<code>usart_hardware_flow_rts_config</code>
Function prototype	<code>void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);</code>
Function descriptions	configure hardware flow control RTS
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart 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 */
```

```
usart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

`usart_hardware_flow_cts_config`

The description of `usart_hardware_flow_cts_config` is shown as below:

Table 3-893. Function `usart_hardware_flow_cts_config`

Function name	<code>usart_hardware_flow_cts_config</code>
Function prototype	<code>void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);</code>
Function descriptions	configure hardware flow control CTS
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
ctsconfig	enable or disable CTS
<i>USART_CTS_ENABLE</i>	enable CTS
<i>USART_CTS_DISABLE</i>	disable CTS

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 hardware flow control CTS */
```

```
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

usart_dma_receive_config

The description of usart_dma_receive_config is shown as below:

Table 3-894. Function usart_dma_receive_config

Function name	usart_dma_receive_config
Function prototype	void usart_dma_receive_config(uint32_t usart_periph, uint8_t dmaconfig);
Function descriptions	configure USART DMA reception
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
dmaconfig	USART DMA mode
USART_RECEIVE_DMA_ENABLE	enable USART DMA for reception
USART_RECEIVE_DMA_DISABLE	disable USART DMA for reception
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 DMA for reception */
```

```
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

usart_dma_transmit_config

The description of usart_dma_transmit_config is shown as below:

Table 3-895. Function usart_dma_transmit_config

Function name	usart_dma_transmit_config
---------------	---------------------------

Function prototype	void usart_dma_transmit_config (uint32_t usart_periph, uint8_t dmaconfig);
Function descriptions	configure USART DMA transmission
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
dmaconfig	USART DMA mode
<i>USART_TRANSMIT_DMA_ENABLE</i>	enable USART DMA for transmission
<i>USART_TRANSMIT_DMA_DISABLE</i>	disable USART DMA for transmission
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 DMA for transmission */
```

```
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

usart_flag_get

The description of usart_flag_get is shown as below:

Table 3-896. Function usart_flag_get

Function name	usart_flag_get
Function prototype	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
Function descriptions	get flag in STAT0/STAT1 register
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart 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 Table 3-848. Enum usart_flag_enum.
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get flag USART0 state */

FlagStatus status;

status = usart_flag_get(USART0, USART_FLAG_TBE);
```

usart_flag_clear

The description of usart_flag_clear is shown as below:

Table 3-897. Function usart_flag_clear

Function name	usart_flag_clear
Function prototype	void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);
Function descriptions	clear flag in STAT0/STAT1 register
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
flag	USART flags, refer to Table 3-848. Enum usart_flag_enum .
USART_FLAG_CTS	CTS change flag
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_TC	transmission complete
USART_FLAG_RBNE	read data buffer not empty
USART_FLAG_EB	end of block flag
USART_FLAG_RT	receiver timeout flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear USART0 flag */

usart_flag_clear(USART0, USART_FLAG_TC);
```

usart_interrupt_enable

The description of usart_interrupt_enable is shown as below:

Table 3-898. Function usart_interrupt_enable

Function name	usart_interrupt_enable
----------------------	------------------------

Function prototype	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
Function descriptions	enable USART interrupt
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
interrupt	USART interrupt, refer to Table 3-850. Enum usart_interrupt_enum .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 TBE interrupt */
```

```
usart_interrupt_enable(USART0, USART_INT_TBE);
```

usart_interrupt_disable

The description of usart_interrupt_disable is shown as below:

Table 3-899. Function usart_interrupt_disable

Function name	usart_interrupt_disable
Function prototype	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
Function descriptions	disable USART interrupt
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
interrupt	USART interrupt, refer to Table 3-850. Enum usart_interrupt_enum .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 TBE interrupt */
```

```
usart_interrupt_disable(USART0, USART_INT_TBE);
```

usart_interrupt_flag_get

The description of usart_interrupt_flag_get is shown as below:

Table 3-900. Function usart_interrupt_flag_get

Function name	usart_interrupt_flag_get
Function prototype	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
Function descriptions	get USART interrupt and flag status
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
int_flag	USART interrupt flag, refer to Table 3-849. Enum usart_interrupt_flag_enum .
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 3-901. 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	usart 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 Table 3-849. Enum <i>usart_interrupt_flag</i> enum.
<i>USART_INT_FLAG_CTS</i>	CTS interrupt and flag
<i>USART_INT_FLAG_LBD</i>	LIN break detected interrupt and flag
<i>USART_INT_FLAG_TC</i>	transmission complete interrupt and flag
<i>USART_INT_FLAG_RBNE</i>	read data buffer not empty interrupt and flag
<i>USART_INT_FLAG_EB</i>	interrupt enable bit of end of block event and flag
<i>USART_INT_FLAG_RT</i>	interrupt enable bit of receive timeout event and flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the USART0 interrupt flag status */
```

```
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

3.29. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.29.1](#), the WWDGT firmware functions are introduced in chapter [3.29.2](#).

3.29.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

Table 3-902. WWDGT Registers

Registers	Descriptions
WWDGT_CTL	control register
WWDGT_CFG	configuration register
WWDGT_STAT	status register

3.29.2. Descriptions of Peripheral functions

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

Table 3-903. WWDGT firmware function

Function name	Function description
wwdgt_deinit	reset the window watchdog timer configuration
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_flag_get	check early wakeup interrupt state of WWDGT
wwdgt_flag_clear	clear early wakeup interrupt state of WWDGT
wwdgt_interrupt_enable	enable early wakeup interrupt of WWDGT
wwdgt_interrupt_flag_get	get early wakeup interrupt flag of WWDGT

wwdgt_deinit

The description of wwdgt_deinit is shown as below:

Table 3-904. 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	rcu_periph_reset_enable / rcu_periph_reset_disable
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 3-905. 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 3-906. 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 3-907. 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</i> <i>IV1</i>	the time base of window watchdog counter = (PCLK1/4096)/1
<i>WWDGT_CFG_PSC_D</i> <i>IV2</i>	the time base of window watchdog counter = (PCLK1/4096)/2
<i>WWDGT_CFG_PSC_D</i> <i>IV4</i>	the time base of window watchdog counter = (PCLK1/4096)/4
<i>WWDGT_CFG_PSC_D</i> <i>IV8</i>	the time base of window watchdog counter = (PCLK1/4096)/8
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

wwdgt_flag_get

The description of wwdgt_flag_get is shown as below:

Table 3-908. 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 reaches 0x40 or refreshes before it reaches the window value */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get ( );
```

wwdgt_flag_clear

The description of wwdgt_flag_clear is shown as below:

Table 3-909. 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( );
```

wwdgt_interrupt_enable

The description of wwdgt_interrupt_enable is shown as below:

Table 3-910. 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_interrupt_flag_get

The description of wwdgt_interrupt_flag_get is shown as below:

Table 3-911. Function wwdgt_interrupt_flag_get

Function name	wwdgt_interrupt_flag_get
Function prototype	FlagStatus wwdgt_interrupt_flag_get(void);
Function descriptions	get early wakeup interrupt flag of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* test if the counter reaches 0x40 or refreshes before it reaches the window value */
```

```
FlagStatus status;
```

```
status = wwdgt_interrupt_flag_get ( );
```

4. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Oct.31, 2018
1.1	Revision contents: gd32f20x has no usbd, the usbd chapter is deleted from the catalog; the folder description in section 2.1.2 is modified; the folder structure diagram in section 2.1.1 is replaced; the folder description in section 2.1.4 is modified; the header format of some drawings is modified	Sep.30, 2020
1.2	Rebase Release	Oct.31, 2021
1.3	<ol style="list-style-type: none"> 1. Add function exmc_sdram_struct_command_para_init() 2. Change function void pmu_to_standbymode(uint8_t standbymodecmd) 3. Change all function qspi_xxx() to spi_quad_xxx() 4. Add function fwdgt_prescaler_value_config() fwdgt_reload_value_config() 5. Delete USBFS chapter 	Jul.31, 2022
1.4	<ol style="list-style-type: none"> 1. Change function usart_dma_enable() and usart_dma_disable() to usart_dma_receive_config() and usart_dma_transmit_config() 2. Change function tli_ckey_init() to tli_color_key_init() 3. Add function tli_struct_para_init() tli_layer_struct_para_init() tli_lut_struct_para_init() 	Jun.30, 2023
1.5	The entire DAC chapter has been modified	Dec.31, 2023
1.6	Update the can_transmission_stop() function	Feb.6, 2026

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 according to the laws of the People's Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. 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.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively "Unintended Uses"). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby release the Company as well as its suppliers and/or distributors from, any claim, damage, or other liability arising from or related to all Unintended Uses of the Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates as well as its 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 Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.