

GigaDevice Semiconductor Inc.

GD32E50x
Arm® Cortex®-M33 32-bit MCU

Firmware Library
User Guide

Revison 1.5

(Dec. 2022)

Table of Contents

Table of Contents	2
List of Figures	5
List of Tables	6
1. Introduction	31
1.1. Rules of User Manual and Firmware Library	31
1.1.1. Peripherals	31
1.1.2. Naming rules	32
2. Firmware Library Overview	34
2.1. File Structure of Firmware Library	34
2.1.1. Examples Folder	35
2.1.2. Firmware Folder	35
2.1.3. Template Folder	36
2.1.4. Utilities Folder	38
2.2. File descriptions of Firmware Library	39
3. Firmware Library of Standard Peripherals	40
3.1. Overview of Firmware Library of Standard Peripherals	40
3.2. ADC	40
3.2.1. Descriptions of Peripheral registers	40
3.2.2. Descriptions of Peripheral functions	41
3.3. BKP	74
3.3.1. Descriptions of Peripheral registers	75
3.3.2. Descriptions of Peripheral functions	75
3.4. CAN	87
3.4.1. Descriptions of Peripheral registers	87
3.4.2. Descriptions of Peripheral functions	89
3.5. CRC	115
3.5.1. Descriptions of Peripheral registers	115
3.5.2. Descriptions of Peripheral functions	116
3.6. CTC	123
3.6.1. Descriptions of Peripheral registers	123
3.6.2. Descriptions of Peripheral functions	124
3.7. CMP	136
3.7.1. Descriptions of Peripheral registers	136
3.7.2. Descriptions of Peripheral functions	137

3.8. DAC	143
3.8.1. Descriptions of Peripheral registers	143
3.8.2. Descriptions of Peripheral functions	144
3.9. DBG	165
3.9.1. Descriptions of Peripheral registers	165
3.9.2. Descriptions of Peripheral functions	166
3.10. DMA	171
3.10.1. Descriptions of Peripheral registers	171
3.10.2. Descriptions of Peripheral functions	172
3.11. ENET	192
3.11.1. Descriptions of Peripheral registers	192
3.11.2. Descriptions of Peripheral functions	194
3.12. EXMC	282
3.12.1. Descriptions of Peripheral registers	282
3.12.2. Descriptions of Peripheral functions	283
3.13. EXTI	302
3.13.1. Descriptions of Peripheral registers	302
3.13.2. Descriptions of Peripheral functions	302
3.14. FMC	310
3.14.1. Descriptions of Peripheral registers	310
3.14.2. Descriptions of Peripheral functions	310
3.15. FWDGT	328
3.15.1. Descriptions of Peripheral registers	329
3.15.2. Descriptions of Peripheral functions	329
3.16. GPIO	334
3.16.1. Descriptions of Peripheral registers	334
3.16.2. Descriptions of Peripheral functions	334
3.17. SHRTIMER	353
3.17.1. Descriptions of Peripheral registers	353
3.17.2. Descriptions of Peripheral functions	355
3.18. I2C	421
3.18.1. Descriptions of Peripheral registers	422
3.18.2. Descriptions of Peripheral functions	422
3.19. MISC	485
3.19.1. Descriptions of Peripheral registers	485
3.19.2. Descriptions of Peripheral functions	485
3.20. PMU	493
3.20.1. Descriptions of Peripheral registers	493
3.20.2. Descriptions of Peripheral functions	493

3.21. RCU	506
3.21.1. Descriptions of Peripheral registers	506
3.21.2. Descriptions of Peripheral functions	507
3.22. RTC	545
3.22.1. Descriptions of Peripheral registers	545
3.22.2. Descriptions of Peripheral functions	546
3.23. SDIO	553
3.23.1. Descriptions of Peripheral registers	553
3.23.2. Descriptions of Peripheral functions	554
3.24. SPI	584
3.24.1. Descriptions of Peripheral registers	584
3.24.2. Descriptions of Peripheral functions	585
3.25. SQPI	612
3.25.1. Descriptions of Peripheral registers	612
3.25.2. Descriptions of Peripheral functions	612
3.26. TIMER	618
3.26.1. Descriptions of Peripheral registers	619
3.26.2. Descriptions of Peripheral functions	619
3.27. TMU	675
3.27.1. Descriptions of Peripheral registers	675
3.27.2. Descriptions of Peripheral functions	675
3.28. USART	681
3.28.1. Descriptions of Peripheral registers	681
3.28.2. Descriptions of Peripheral functions	682
3.29. WWDGT	736
3.29.1. Descriptions of Peripheral registers	737
3.29.2. Descriptions of Peripheral functions	737
4. Revision history	742

List of Figures

Figure 2-1. File structure of firmware library of GD32E50x	34
Figure 2-2. Select peripheral example files	36
Figure 2-3. Copy the peripheral example files	37
Figure 2-4. Open the project file	37
Figure 2-5. Configure project files	38
Figure 2-6. Compile-debug-download	38

List of Tables

Table 1-1. Peripherals	31
Table 2-1. Function descriptions of Firmware Library.....	39
Table 3-1. Peripheral function format of Firmware Library.....	40
Table 3-2. ADC Registers	40
Table 3-3. ADC firmware function.....	41
Table 3-4. Function adc_deinit.....	42
Table 3-5. Function adc_enable	43
Table 3-6. Function adc_disable	43
Table 3-7. Function adc_calibration_enable	44
Table 3-8. Function adc_calibration_number	44
Table 3-9. Function adc_dma_mode_enable	45
Table 3-10. Function adc_dma_mode_disable	46
Table 3-11. Function adc_tempsensor_vrefint_enable	46
Table 3-12. Function adc_tempsensor_vrefint_disable.....	47
Table 3-13. Function adc_discontinuous_mode_config	47
Table 3-14. Function adc_mode_config	48
Table 3-15. Function adc_special_function_config	49
Table 3-16. Function adc_data_alignment_config.....	50
Table 3-17. Function adc_channel_length_config.....	50
Table 3-18. Function adc_regular_channel_config	51
Table 3-19. Function adc_inserted_channel_config	52
Table 3-20. Function adc_inserted_channel_offset_config	53
Table 3-21. Function adc_channel_differential_mode_config	54
Table 3-22. Function adc_external_trigger_config	55
Table 3-23. Function adc_external_trigger_source_config	56
Table 3-24. Function adc_software_trigger_enable	58
Table 3-25. Function adc_regular_data_read	59
Table 3-26. Function adc_inserted_data_read	59
Table 3-27. Function adc_sync_mode_convert_value_read	60
Table 3-28. Function adc_watchdog0_single_channel_enable	60
Table 3-29. Function adc_watchdog0_group_channel_enable	61
Table 3-30. Function adc_watchdog0_disable	62
Table 3-31. Function adc_watchdog1_channel_config	62
Table 3-32. Function adc_watchdog2_channel_config	63
Table 3-33. Function adc_watchdog1_disable	64
Table 3-34. Function adc_watchdog2_disable	64
Table 3-35. Function adc_watchdog0_threshold_config	65
Table 3-36. Function adc_watchdog1_threshold_config	65
Table 3-37. Function adc_watchdog2_threshold_config	66
Table 3-38. Function adc_resolution_config	67

Table 3-39. Function adc_oversample_mode_config	67
Table 3-40. Function adc_oversample_mode_enable	69
Table 3-41. Function adc_oversample_mode_disable	70
Table 3-42. Function adc_flag_get	70
Table 3-43. Function adc_flag_clear	71
Table 3-44. Function adc_interrupt_enable	72
Table 3-45. Function adc_interrupt_disable	72
Table 3-46. Function adc_interrupt_flag_get	73
Table 3-47. Function adc_interrupt_flag_clear	74
Table 3-48. BKP Registers	75
Table 3-49. BKP firmware function	75
Table 3-50. Enum bkp_data_register_enum	76
Table 3-51. Function bkp_deinit	77
Table 3-52. Function bkp_write_data	77
Table 3-53. Function bkp_data_read	78
Table 3-54. Function bkp_rtc_calibration_output_enable	78
Table 3-55. Function bkp_rtc_calibration_output_disable	79
Table 3-56. Function bkp_rtc_signal_output_enable	79
Table 3-57. Function bkp_rtc_signal_output_disable	80
Table 3-58. Function bkp_rtc_output_select	80
Table 3-59. Function bkp_rtc_clock_output_select	81
Table 3-60. Function bkp_rtc_clock_calibration_direction	81
Table 3-61. Function bkp_rtc_calibration_value_set	82
Table 3-62. Function bkp_tamper_detection_enable	83
Table 3-63. Function bkp_tamper_detection_disable	83
Table 3-64. Function bkp_tamper_active_level_set	83
Table 3-65. Function bkp_tamper_interrupt_enable	84
Table 3-66. Function bkp_tamper_interrupt_disable	85
Table 3-67. Function bkp_flag_get	85
Table 3-68. Function bkp_flag_clear	86
Table 3-69. Function bkp_interrupt_flag_get	86
Table 3-70. Function bkp_interrupt_flag_clear	87
Table 3-71. CAN Registers	87
Table 3-72. CAN-FD Registers	88
Table 3-73. CAN firmware function	89
Table 3-74. can_parameter_struct	90
Table 3-75. can_transmit_message_struct	90
Table 3-76. can_transmit_message_struct (for CAN-FD)	90
Table 3-77. can_receive_message_struct	91
Table 3-78. can_receive_message_struct (for CAN-FD)	91
Table 3-79. can_fd_tdc_struct (for CAN-FD)	91
Table 3-80. can_fdframe_struct (for CAN-FD)	91
Table 3-81. can_filter_parameter_struct	92
Table 3-82. Enum can_interrupt_flag_enum	92

Table 3-83. Enum can_flag_enum	93
Table 3-84. Enum can_error_enum	94
Table 3-85. Enum can_transmit_state_enum	94
Table 3-86. Enum can_format_fifo_enum	94
Table 3-87. Enum can_struct_type_enum.....	94
Table 3-88. Function can_deinit.....	95
Table 3-89. Function can_struct_para_init	95
Table 3-90. Function can_init	96
Table 3-91. Function can_filter_init.....	96
Table 3-92. Function can_filter_mask_mode_init.....	97
Table 3-93. Function can_monitor_mode_set	98
Table 3-94. Function can_fd_init	98
Table 3-95. Function can_fd_function_enable	99
Table 3-96. Function can_fd_function_disable	100
Table 3-97. Function can1_filter_start_bank	100
Table 3-98. Function can_debug_freeze_enable	101
Table 3-99. Function can_debug_freeze_disable	101
Table 3-100. Function can_time_trigger_mode_enable.....	102
Table 3-101. Function can_time_trigger_mode_disable	102
Table 3-102. Function can_message_transmit	103
Table 3-103. Function can_transmit_states	104
Table 3-104. Function can_transmission_stop	104
Table 3-105. Function can_message_receive	105
Table 3-106. Function can_fifo_release.....	106
Table 3-107. Function can_receive_message_length_get	106
Table 3-108. Function can_working_mode_set	107
Table 3-109. Function can_wakeup.....	107
Table 3-110. Function can_error_get	108
Table 3-111. Function can_receive_error_number_get	108
Table 3-112. Function can_transmit_error_number_get	109
Table 3-113. Function can_interrupt_enable	109
Table 3-114. Function can_interrupt_disable	110
Table 3-115. Function can_flag_get	111
Table 3-116. Function can_flag_clear	113
Table 3-117. Function can_interrupt_flag_get.....	113
Table 3-118. Function can_interrupt_flag_clear.....	115
Table 3-119. CRC Registers	116
Table 3-120. CRC firmware function	116
Table 3-121. Function crc_deinit	116
Table 3-122. Function crc_data_register_reset.....	117
Table 3-123. Function crc_reverse_output_data_enable	117
Table 3-124. Function crc_reverse_output_data_disable	118
Table 3-125. Function crc_input_data_reverse_config	118
Table 3-126. Function crc_data_register_read.....	119

Table 3-127. Function <code>crc_free_data_register_read</code>	119
Table 3-128. Function <code>crc_free_data_register_write</code>	120
Table 3-129. Function <code>crc_init_data_register_write</code>	120
Table 3-130. Function <code>crc_polynomial_size_set</code>	121
Table 3-131. Function <code>crc_polynomial_set</code>	121
Table 3-132. Function <code>crc_single_data_calculate</code>	122
Table 3-133. Function <code>crc_block_data_calculate</code>	122
Table 3-134. CTC Registers	123
Table 3-135. CTC firmware function.....	124
Table 3-136. Function <code>ctc_deinit</code>	124
Table 3-137. Function <code>ctc_counter_enable</code>	125
Table 3-138. Function <code>ctc_counter_disable</code>	125
Table 3-139. Function <code>ctc irc48m_trim_value_config</code>	126
Table 3-140. Function <code>ctc_software_refresource_pulse_generate</code>	126
Table 3-141. Function <code>ctc_hardware_trim_mode_config</code>	127
Table 3-142. Function <code>ctc_refresource_polarity_config</code>	127
Table 3-143. Function <code>ctc_refresource_signal_select</code>	128
Table 3-144. Function <code>ctc_refresource_prescaler_config</code>	128
Table 3-145. Function <code>ctc_clock_limit_value_config</code>	129
Table 3-146. Function <code>ctc_counter_reload_value_config</code>	130
Table 3-147. Function <code>ctc_counter_capture_value_read</code>	130
Table 3-148. Function <code>ctc_counter_direction_read</code>	131
Table 3-149. Function <code>ctc_counter_reload_value_read</code>	131
Table 3-150. Function <code>ctc irc48m_trim_value_read</code>	132
Table 3-151. Function <code>ctc_flag_get</code>	132
Table 3-152. Function <code>ctc_flag_clear</code>	133
Table 3-153. Function <code>ctc_interrupt_enable</code>	134
Table 3-154. Function <code>ctc_interrupt_disable</code>	134
Table 3-155. Function <code>ctc_interrupt_flag_get</code>	135
Table 3-156. Function <code>ctc_interrupt_flag_clear</code>	136
Table 3-157. CMP Registers.....	137
Table 3-158. CMP firmware function	137
Table 3-159. Enum <code>cmp_enum</code>	137
Table 3-160. Enum <code>inverting_input_enum</code>	137
Table 3-161. Enum <code>cmp_output_enum</code>	138
Table 3-162. Enum <code>cmp_outputblank_enum</code>	138
Table 3-163. Function <code>cmp_deinit</code>	138
Table 3-164. Function <code>cmp_input_init</code>	139
Table 3-165. Function <code>cmp_input_init</code>	139
Table 3-166. Function <code>cmp_outputblank_init</code>	140
Table 3-167. Function <code>cmp_enable</code>	141
Table 3-168. Function <code>cmp_disable</code>	142
Table 3-169. Function <code>cmp_lock_enable</code>	142
Table 3-170. Function <code>cmp_output_level_get</code>	143

Table 3-171. DAC Registers	143
Table 3-172. DAC firmware function	144
Table 3-173. Function <code>dac_deinit</code>	145
Table 3-174. Function <code>dac_enable</code>	145
Table 3-175. Function <code>dac_disable</code>	146
Table 3-176. Function <code>dac_dma_enable</code>	146
Table 3-177. Function <code>dac_dma_disable</code>	147
Table 3-178. Function <code>dac_output_buffer_enable</code>	147
Table 3-179. Function <code>dac_output_buffer_disable</code>	148
Table 3-180. Function <code>dac_output_value_get</code>	148
Table 3-181. Function <code>dac_data_set</code>	149
Table 3-182. Function <code>dac_output_buffer_enable</code>	150
Table 3-183. Function <code>dac_output_fifo_disable</code>	150
Table 3-184. Function <code>dac_output_fifo_number_get</code>	151
Table 3-185. Function <code>dac_trigger_enable</code>	151
Table 3-186. Function <code>dac_trigger_disable</code>	152
Table 3-187. Function <code>dac_trigger_source_config</code>	152
Table 3-188. Function <code>dac_software_trigger_enable</code>	153
Table 3-189. Function <code>dac_software_trigger_disable</code>	154
Table 3-190. Function <code>dac_wave_mode_config</code>	154
Table 3-191. Function <code>dac_wave_bit_width_config</code>	155
Table 3-192. Function <code>dac_lfsr_noise_config</code>	156
Table 3-193. Function <code>dac_triangle_noise_config</code>	156
Table 3-194. Function <code>dac_concurrent_enable</code>	157
Table 3-195. Function <code>dac_concurrent_disable</code>	157
Table 3-196. Function <code>dac_concurrent_software_trigger_enable</code>	158
Table 3-197. Function <code>dac_concurrent_software_trigger_disable</code>	158
Table 3-198. Function <code>dac_concurrent_output_buffer_enable</code>	159
Table 3-199. Function <code>dac_concurrent_output_buffer_disable</code>	159
Table 3-200. Function <code>dac_concurrent_data_set</code>	160
Table 3-201. Function <code>dac_flag_get</code>	160
Table 3-202. Function <code>dac_flag_clear</code>	161
Table 3-203. Function <code>dac_interrupt_enable</code>	162
Table 3-204. Function <code>dac_interrupt_enable</code>	163
Table 3-205. Function <code>dac_interrupt_flag_get</code>	164
Table 3-206. Function <code>dac_interrupt_flag_clear</code>	164
Table 3-207. DBG Registers.....	165
Table 3-208. DBG firmware function	166
Table 3-209. Enum <code>dbg_periph_enum</code>	166
Table 3-210. Function <code>dbg_deinit</code>	167
Table 3-211. Function <code>dbg_id_get</code>	167
Table 3-212. Function <code>dbg_low_power_enable</code>	168
Table 3-213. Function <code>dbg_low_power_disable</code>	168
Table 3-214. Function <code>dbg_periph_enable</code>	169

Table 3-215. Function <code>dbg_periph_disable</code>	170
Table 3-216. Function <code>dbg_trace_pin_enable</code>	170
Table 3-217. Function <code>dbg_trace_pin_disable</code>	171
Table 3-218. DMA Registers.....	171
Table 3-219. DMA firmware function	172
Table 3-220. Enum <code>dma_channel_enum</code>	173
Table 3-221. Structure <code>dma_parameter_struct</code>	173
Table 3-222. Function <code>dma_deinit</code>	173
Table 3-223. Function <code>dma_struct_para_init</code>	174
Table 3-224. Function <code>dma_init</code>	174
Table 3-225. Function <code>dma_circulation_enable</code>	175
Table 3-226. Function <code>dma_circulation_disable</code>	176
Table 3-227. Function <code>dma_memory_to_memory_enable</code>	177
Table 3-228. Function <code>dma_memory_to_memory_disable</code>	177
Table 3-229. Function <code>dma_channel_enable</code>	178
Table 3-230. Function <code>dma_channel_disable</code>	178
Table 3-231. Function <code>dma_periph_address_config</code>	179
Table 3-232. Function <code>dma_memory_address_config</code>	180
Table 3-233. Function <code>dma_transfer_number_config</code>	180
Table 3-234. Function <code>dma_transfer_number_get</code>	181
Table 3-235. Function <code>dma_priority_config</code>	182
Table 3-236. Function <code>dma_memory_width_config</code>	182
Table 3-237. Function <code>dma_periph_width_config</code>	183
Table 3-238. Function <code>dma_memory_increase_enable</code>	184
Table 3-239. Function <code>dma_memory_increase_disable</code>	185
Table 3-240. Function <code>dma_periph_increase_enable</code>	185
Table 3-241. Function <code>dma_periph_increase_disable</code>	186
Table 3-242. Function <code>dma_transfer_direction_config</code>	186
Table 3-243. Function <code>dma_flag_get</code>	187
Table 3-244. Function <code>dma_flag_clear</code>	188
Table 3-245. Function <code>dma_interrupt_enable</code>	189
Table 3-246. Function <code>dma_interrupt_disable</code>	189
Table 3-247. Function <code>dma_interrupt_flag_get</code>	190
Table 3-248. Function <code>dma_interrupt_flag_clear</code>	191
Table 3-249. ENET Registers	192
Table 3-250. ENET firmware function	194
Table 3-251. Structure <code>enet_initpara_struct</code>	197
Table 3-252. Structure <code>enet_descriptors_struct</code>	198
Table 3-253. Structure <code>enet_ptp_systime_struct</code>	198
Table 3-254. Enum <code>enet_flag_enum</code>	198
Table 3-255. Enum <code>enet_flag_clear_enum</code>	200
Table 3-256. Enum <code>enet_int_enum</code>	201
Table 3-257. Enum <code>enet_int_flag_enum</code>	202
Table 3-258. Enum <code>enet_int_flag_clear_enum</code>	204

Table 3-259. Enum enet_desc_reg_enum	205
Table 3-260. Enum enet_msc_counter_enum	205
Table 3-261. Enum enet_option_enum	205
Table 3-262. Enum enet_mediamode_enum	206
Table 3-263. Enum enet_chksumconf_enum	206
Table 3-264. Enum enet_frmrecept_enum	207
Table 3-265. Enum enet_registers_type_enum	207
Table 3-266. Enum enet_dmadirection_enum	207
Table 3-267. Enum enet_phydirection_enum	207
Table 3-268. Enum enet_regdirection_enum	207
Table 3-269. Enum enet_macaddress_enum	208
Table 3-270. Enum enet_descstate_enum	208
Table 3-271. Enum enet_msc_preset_enum	208
Table 3-272. Function enet_deinit	209
Table 3-273. Function enet_initpara_config	209
Table 3-274. Function enet_init	212
Table 3-275. Function enet_software_reset	214
Table 3-276. Function enet_rxframe_size_get	214
Table 3-277. Function enet_descriptors_chain_init	215
Table 3-278. Function enet_descriptors_ring_init	216
Table 3-279. Function enet_frame_receive	216
Table 3-280. Function enet_frame_transmit	217
Table 3-281. Function enet_transmit_checksum_config	217
Table 3-282. Function enet_enable	218
Table 3-283. Function enet_disable	219
Table 3-284. Function enet_mac_address_set	219
Table 3-285. Function enet_mac_address_get	220
Table 3-286. Function enet_flag_get	221
Table 3-287. Function enet_flag_clear	223
Table 3-288. Function enet_interrupt_enable	224
Table 3-289. Function enet_interrupt_disable	226
Table 3-290. Function enet_interrupt_flag_get	227
Table 3-291. Function enet_interrupt_flag_clear	229
Table 3-292. Function enet_tx_enable	230
Table 3-293. Function enet_tx_disable	231
Table 3-294. Function enet_rx_enable	231
Table 3-295. Function enet_rx_disable	232
Table 3-296. Function enet_registers_get	232
Table 3-297. Function enet_debug_status_get	233
Table 3-298. Function enet_address_filter_enable	234
Table 3-299. Function enet_address_filter_disable	235
Table 3-300. Function enet_address_filter_config	235
Table 3-301. Function enet_phy_config	237
Table 3-302. Function enet_phy_write_read	237

Table 3-303. Function enet_phyloopback_enable	238
Table 3-304. Function enet_phyloopback_disable	238
Table 3-305. Function enet_forward_feature_enable	239
Table 3-306. Function enet_forward_feature_disable	240
Table 3-307. Function enet_fliter_feature_enable	240
Table 3-308. Function enet_fliter_feature_disable	241
Table 3-309. Function enet_pauseframe_generate	242
Table 3-310. Function enet_pauseframe_detect_config	242
Table 3-311. Function enet_pauseframe_config	243
Table 3-312. Function enet_flowcontrol_threshold_config	244
Table 3-313. Function enet_flowcontrol_feature_enable	245
Table 3-314. Function enet_flowcontrol_feature_disable	246
Table 3-315. Function enet_dmaprocess_state_get	247
Table 3-316. Function enet_dmaprocess_resume	247
Table 3-317. Function enet_rxprocess_check_recovery	248
Table 3-318. Function enet_txfifo_flush	249
Table 3-319. Function enet_current_desc_address_get	249
Table 3-320. Function enet_desc_information_get	250
Table 3-321. Function enet_missed_frame_counter_get	251
Table 3-322. Function enet_desc_flag_get	251
Table 3-323. Function enet_desc_flag_set	253
Table 3-324. Function enet_desc_flag_clear	254
Table 3-325. Function enet_rx_desc_immediate_receive_complete_interrupt	255
Table 3-326. Function enet_rx_desc_delay_receive_complete_interrupt	256
Table 3-327. Function enet_rxframe_drop	256
Table 3-328. Function enet_dma_feature_enable	257
Table 3-329. Function enet_dma_feature_disable	257
Table 3-330. Function enet_rx_desc_enhanced_status_get	258
Table 3-331. Function enet_desc_select_enhanced_mode	259
Table 3-332. Function enet_ptp_enhanced_descriptors_chain_init	259
Table 3-333. Function enet_ptp_enhanced_descriptors_ring_init	260
Table 3-334. Function enet_ptpframe_receive_enhanced_mode	261
Table 3-335. Function enet_ptpframe_transmit_enhanced_mode	261
Table 3-336. Function enet_desc_select_normal_mode	262
Table 3-337. Function enet_ptp_normal_descriptors_chain_init	263
Table 3-338. Function enet_ptp_normal_descriptors_ring_init	263
Table 3-339. Function enet_ptpframe_receive_normal_mode	264
Table 3-340. Function enet_ptpframe_transmit_normal_mode	265
Table 3-341. Function enet_wum_filter_register_pointer_reset	265
Table 3-342. Function enet_wum_filter_config	266
Table 3-343. Function enet_wum_feature_enable	266
Table 3-344. Function enet_wum_feature_disable	267
Table 3-345. Function enet_msc_counters_reset	268
Table 3-346. Function enet_msc_feature_enable	268

Table 3-347. Function enet_msc_feature_disable	269
Table 3-348. Function enet_msc_counters_preset_config	269
Table 3-349. Function enet_msc_counters_get	270
Table 3-350. Function enet_ptp_subsecond_2_nanosecond	271
Table 3-351. Function enet_ptp_nanosecond_2_subsecond	271
Table 3-352. Function enet_ptp_feature_enable	272
Table 3-353. Function enet_ptp_feature_disable	273
Table 3-354. Function enet_ptp_timestamp_function_config	274
Table 3-355. Function enet_ptp_subsecond_increment_config	275
Table 3-356. Function enet_ptp_timestamp_addend_config	276
Table 3-357. Function enet_ptp_timestamp_update_config	276
Table 3-358. Function enet_ptp_expected_time_config	277
Table 3-359. Function enet_ptp_system_time_get	277
Table 3-360. enet_ptp_pps_output_frequency_config	278
Table 3-361. enet_ptp_start	279
Table 3-362. enet_ptp_finecorrection_adjfreq	280
Table 3-363. enet_ptp_coarsecorrection_systime_update	280
Table 3-364. enet_ptp_finecorrection_settime	281
Table 3-365. enet_ptp_flag_get	281
Table 3-366. Function enet_initpara_reset	282
Table 3-367. EXMC Registers	283
Table 3-368. EXMC firmware function	283
Table 3-369. Structure exmc_norsram_timing_parameter_struct	284
Table 3-370. Structure exmc_norsram_parameter_struct	284
Table 3-371. Structure exmc_nand_pccard_timing_parameter_struct	285
Table 3-372. Structure exmc_nand_parameter_struct	285
Table 3-373. Structure exmc_pccard_parameter_struct	285
Table 3-374. Function exmc_norsram_deinit	286
Table 3-375. Function exmc_norsram_struct_para_init	286
Table 3-376. Function exmc_norsram_init	287
Table 3-377. Function exmc_norsram_enable	288
Table 3-378. Function exmc_norsram_disable	289
Table 3-379. Function exmc_norsram_page_size_config	289
Table 3-380. Function exmc_nand_deinit	290
Table 3-381. Function exmc_nand_struct_para_init	290
Table 3-382. Function exmc_nand_init	291
Table 3-383. Function exmc_nand_enable	292
Table 3-384. Function exmc_nand_disable	293
Table 3-385. Function exmc_nand_ecc_config	293
Table 3-386. Function exmc_ecc_get	294
Table 3-387. Function exmc_pccard_deinit	294
Table 3-388. Function exmc_pccard_struct_para_init	295
Table 3-389. Function exmc_pccard_init	295
Table 3-390. Function exmc_pccard_enable	296

Table 3-391. Function exmc_pccard_disable	297
Table 3-392. Function exmc_interrupt_enable	297
Table 3-393. Function exmc_interrupt_disable	298
Table 3-394. Function exmc_flag_get	299
Table 3-395. Function exmc_flag_clear	299
Table 3-396. Function exmc_interrupt_flag_get	300
Table 3-397. Function exmc_interrupt_flag_clear	301
Table 3-398. EXTI Registers	302
Table 3-399. EXTI firmware function	302
Table 3-400. Enum exti_line_enum	303
Table 3-401. Enum exti_mode_enum	303
Table 3-402. Enum exti_trig_type_enum	303
Table 3-403. Function exti_deinit	304
Table 3-404. Function exti_init	304
Table 3-405. Function exti_interrupt_enable	305
Table 3-406. Function exti_interrupt_disable	305
Table 3-407. Function exti_event_enable	306
Table 3-408. Function exti_event_disable	306
Table 3-409. Function exti_software_interrupt_enable	307
Table 3-410. Function exti_software_interrupt_disable	307
Table 3-411. Function exti_flag_get	308
Table 3-412. Function exti_flag_clear	308
Table 3-413. Function exti_interrupt_flag_get	309
Table 3-414. Function exti_interrupt_flag_clear	309
Table 3-415. FMC Registers	310
Table 3-416. FMC firmware function	310
Table 3-417. fmc_state_enum	311
Table 3-418. Function fmc_unlock	311
Table 3-419. Function fmc_lock	312
Table 3-420. Function fmc_wscnt_set	312
Table 3-421. Function fmc_prefetch_enable	313
Table 3-422. Function fmc_prefetch_disable	313
Table 3-423. Function fmc_ibus_enable	314
Table 3-424. Function fmc_ibus_disable	314
Table 3-425. Function fmc_ibus_reset	315
Table 3-426. Function fmc_dbus_enable	315
Table 3-427. Function fmc_dbus_disable	316
Table 3-428. Function fmc_dbus_reset	316
Table 3-429. Function fmc_page_erase	317
Table 3-430. Function fmc_mass_erase	317
Table 3-431. Function fmc_word_program	318
Table 3-432. Function ob_unlock	318
Table 3-433. Function ob_lock	319
Table 3-434. Function ob_erase	319

Table 3-435. Function ob_write_protection_enable	320
Table 3-436. Function ob_security_protection_config	321
Table 3-437. Function ob_user_write	321
Table 3-438. Function ob_data_program	322
Table 3-439. Function ob_user_get	323
Table 3-440. Function ob_data_program	323
Table 3-441. Function ob_write_protection_get	324
Table 3-442. Function ob_security_protection_flag_get	324
Table 3-443. Function fmc_flag_get	325
Table 3-444. Function fmc_flag_clear	325
Table 3-445. Function fmc_interrupt_enable	326
Table 3-446. Function fmc_interrupt_disable	326
Table 3-447. Function fmc_interrupt_flag_get	327
Table 3-448. Function fmc_interrupt_flag_clear	328
Table 3-449. FWDGT Registers	329
Table 3-450. FWDGT firmware function	329
Table 3-451. Function fwdgt_write_enable	329
Table 3-452. Function fwdgt_write_disable	330
Table 3-453. Function fwdgt_enable	330
Table 3-454. Function fwdgt_prescaler_value_config	331
Table 3-455. Function fwdgt_reload_value_config	331
Table 3-456. Function fwdgt_config	332
Table 3-457. Function fwdgt_counter_reload	332
Table 3-458. Function fwdgt_flag_get	333
Table 3-459. GPIO Registers	334
Table 3-460. GPIO firmware function	335
Table 3-461. Function gpio_deinit	335
Table 3-462. Function gpio_afio_deinit	336
Table 3-463. Function gpio_init	336
Table 3-464. Function gpio_bit_set	337
Table 3-465. Function gpio_bit_reset	338
Table 3-466. Function gpio_bit_write	338
Table 3-467. Function gpio_port_write	339
Table 3-468. Function gpio_input_bit_get	340
Table 3-469. Function gpio_input_port_get	340
Table 3-470. Function gpio_output_bit_get	341
Table 3-471. Function gpio_output_port_get	341
Table 3-472. Function gpio_pin_remap_config	342
Table 3-473. Function gpio_afio_port_config	344
Table 3-474. Function gpio_ethernet_phy_select	348
Table 3-475. Function gpio_exti_source_select	349
Table 3-476. Function gpio_event_output_config	349
Table 3-477. Function gpio_event_output_enable	350
Table 3-478. Function gpio_event_output_disable	351

Table 3-479. Function gpio_pin_lock	351
Table 3-480. Function gpio_compensation_config	352
Table 3-481. Function gpio_compensation_flag_get	352
Table 3-482. SHRTIMER Register	353
Table 3-483. SHRTIMER firmware function	355
Table 3-484. Structure shrtimer_baseinit_parameter_struct	358
Table 3-485. Structure shrtimer_timerinit_parameter_struct	358
Table 3-486. Structure shrtimer_timercfg_parameter_struct	358
Table 3-487. Structure shrtimer_comparecfg_parameter_struct	359
Table 3-488. Structure shrtimer_exevfilter_parameter_struct	359
Table 3-489. Structure shrtimer_deadtimecfg_parameter_struct	359
Table 3-490. Structure shrtimer_carriersignalcfg_parameter_struct	360
Table 3-491. Structure shrtimer_synccfg_parameter_struct	360
Table 3-492. Structure shrtimer_bunchmode_parameter_struct	360
Table 3-493. Structure shrtimer_exeventcfg_parameter_struct	361
Table 3-494. Structure shrtimer_faultcfg_parameter_struct	361
Table 3-495. Structure shrtimer_adctrigcfg_parameter_struct	361
Table 3-496. Structure shrtimer_channel_outputcfg_parameter_struct	361
Table 3-497. Function shrtimer_deinit	362
Table 3-498. Function shrtimer_dll_calibration_start	362
Table 3-499. Function shrtimer_baseinit_struct_para_init	363
Table 3-500. Function shrtimer_timers_base_init	364
Table 3-501. Function shrtimer_timers_counter_enable	364
Table 3-502. Function shrtimer_timers_counter_enable	365
Table 3-503. Function shrtimer_timers_update_event_enable	366
Table 3-504. Function shrtimer_timers_update_event_enable	366
Table 3-505. Function shrtimer_software_update	367
Table 3-506. Function shrtimer_software_counter_reset	368
Table 3-507. Function shrtimer_timerinit_struct_para_init	368
Table 3-508. Function shrtimer_timers_waveform_init	369
Table 3-509. Function shrtimer_timercfg_struct_para_init	370
Table 3-510. Function shrtimer_slavetimer_waveform_config	371
Table 3-511. Function shrtimer_comparecfg_struct_para_init	372
Table 3-512. Function shrtimer_slavetimer_waveform_compare_config	372
Table 3-513. Function shrtimer_channel_outputcfg_struct_para_init	373
Table 3-514. Function shrtimer_slavetimer_waveform_channel_config	374
Table 3-515. Function shrtimer_slavetimer_waveform_channel_software_request	375
Table 3-516. Function shrtimer_slavetimer_waveform_channel_output_level_get	376
Table 3-517. Function shrtimer_slavetimer_waveform_channel_state_get	377
Table 3-518. Function shrtimer_channel_outputcfg_struct_para_init	378
Table 3-519. Function shrtimer_slavetimer_waveform_channel_software_request	379
Table 3-520. Function shrtimer_channel_outputcfg_struct_para_init	380
Table 3-521. Function shrtimer_slavetimer_carriersignal_config	380
Table 3-522. Function shrtimer_output_channel_enable	381

Table 3-523. Function shrtimer_output_channel_disable	382
Table 3-524. Function shrtimer_slavetimer_waveform_compare_config	382
Table 3-525. Function shrtimer_slavetimer_compare_value_get	383
Table 3-526. Function shrtimer_mastertimer_compare_value_config	384
Table 3-527. Function shrtimer_slavetimer_compare_value_get	385
Table 3-528. Function shrtimer_timers_counter_value_config	385
Table 3-529. Function shrtimer_timers_counter_value_get	386
Table 3-530. Function shrtimer_timers_autoreload_value_config	387
Table 3-531. Function shrtimer_timers_autoreload_value_get	387
Table 3-532. Function shrtimer_timers_repetition_value_config	388
Table 3-533. Function shrtimer_timers_repetition_value_get	389
Table 3-534. Function shrtimer_exevfilter_struct_para_init	390
Table 3-535. Function shrtimer_slavetimer_exevent_filtering_config	390
Table 3-536. Function shrtimer_exeventcfg_struct_para_init	391
Table 3-537. Function shrtimer_exevent_config	392
Table 3-538. Function shrtimer_exevent_prescaler	393
Table 3-539. Function shrtimer_synccfg_struct_para_init	393
Table 3-540. Function shrtimer_synchronization_config	394
Table 3-541. Function shrtimer_faultcfg_struct_para_init	395
Table 3-542. Function shrtimer_fault_config	395
Table 3-543. Function shrtimer_fault_prescaler_config	396
Table 3-544. Function shrtimer_fault_input_enable	397
Table 3-545. Function shrtimer_fault_input_disable	397
Table 3-546. Function shrtimer_timers_dma_enable	398
Table 3-547. Function shrtimer_timers_dma_enable	399
Table 3-548. Function shrtimer_dmamode_config	400
Table 3-549. Function shrtimer_bunchmode_struct_para_init	402
Table 3-550. Function shrtimer_bunchmode_config	402
Table 3-551. Function shrtimer_bunchmode_enable	403
Table 3-552. Function shrtimer_bunchmode_disable	404
Table 3-553. Function shrtimer_bunchmode_flag_get	404
Table 3-554. Function shrtimer_bunchmode_software_start	405
Table 3-555. Function shrtimer_slavetimer_capture_config	405
Table 3-556. Function shrtimer_slavetimer_capture_software	407
Table 3-557. Function shrtimer_slavetimer_capture_value_read	407
Table 3-558. Function shrtimer_adctrigcfg_struct_para_init	408
Table 3-559. Function shrtimer_adc_trigger_config	409
Table 3-560. Function shrtimer_timers_flag_get	410
Table 3-561. Function shrtimer_timers_flag_clear	411
Table 3-562. Function shrtimer_common_flag_get	412
Table 3-563. Function shrtimer_common_flag_clear	413
Table 3-564. Function shrtimer_timers_interrupt_enable	414
Table 3-565. Function shrtimer_timers_interrupt_disable	415
Table 3-566. Function shrtimer_timers_interrupt_flag_get	416

Table 3-567. Function shrtimer_timers_interrupt_flag_clear	417
Table 3-568. Function shrtimer_common_interrupt_enable.....	418
Table 3-569. Function shrtimer_common_interrupt_disable	419
Table 3-570. Function shrtimer_common_interrupt_flag_get	420
Table 3-571. Function shrtimer_common_interrupt_flag_clear	421
Table 3-572. I2C Registers	422
Table 3-573. I2C firmware function.....	422
Table 3-574. i2c_flag_enum	425
Table 3-575. i2c_interrupt_enum	426
Table 3-576. i2c_interrupt_flag_enum	426
Table 3-577. i2c2_interrupt_flag_enum	427
Table 3-578. Function i2c_deinit.....	427
Table 3-579. Function i2c_enable	428
Table 3-580. Function i2c_disable	428
Table 3-581. Function i2c_start_on_bus	429
Table 3-582. Function i2c_stop_on_bus	429
Table 3-583. Function i2c_slave_response_to_gcall_enable	430
Table 3-584. Function i2c_slave_response_to_gcall_disable	430
Table 3-585. Function i2c_stretch_scl_low_enable	431
Table 3-586. Function i2c_stretch_scl_low_disable	431
Table 3-587. Function i2c_data_transmit	432
Table 3-588. Function i2c_data_receive	432
Table 3-589. Function i2c_pec_transfer	433
Table 3-590. Function i2c_pec_enable	433
Table 3-591. Function i2c_pec_disable	434
Table 3-592. Function i2c_pec_value_get.....	434
Table 3-593. Function i2c_clock_config	435
Table 3-594. Function i2c_mode_addr_config	436
Table 3-595. Function i2c_smbus_type_config	437
Table 3-596. Function i2c_ack_config	437
Table 3-597. Function i2c_ackpos_config	438
Table 3-598. Function i2c_master_addressing	438
Table 3-599. Function i2c_dualaddr_enable	439
Table 3-600. Function i2c_dualaddr_disable	440
Table 3-601. Function i2c_dma_config	440
Table 3-602. Function i2c_dma_last_transfer_config	441
Table 3-603. Function i2c_software_reset_config	441
Table 3-604. Function i2c_smbus_alert_config	442
Table 3-605. Function i2c_smbus_arp_config	443
Table 3-606. Function i2c_sam_enable	443
Table 3-607. Function i2c_sam_disable	444
Table 3-608. Function i2c_sam_timeout_enable.....	444
Table 3-609. Function i2c_sam_timeout_disable	445
Table 3-610. Function i2c_start_early_termination_mode_config	445

Table 3-611. Function i2c_timeout_calculation_enable.....	446
Table 3-612. Function i2c_timeout_calculation_disable	446
Table 3-613. Function i2c_record_received_slave_address_enable	447
Table 3-614. Function i2c_record_received_slave_address_disable	447
Table 3-615. Function i2c_address_bit_compare_config	448
Table 3-616. Function i2c_status_clear_enable.....	449
Table 3-617. Function i2c_status_clear_disable.....	449
Table 3-618. Function i2c_start_early_termination_mode_config	450
Table 3-619. Function i2c_flag_get	450
Table 3-620. Function i2c_flag_clear	452
Table 3-621. Function i2c_interrupt_enable	453
Table 3-622. Function i2c_interrupt_disable	453
Table 3-623. Function i2c_interrupt_flag_get.....	454
Table 3-624. Function i2c_interrupt_flag_clear.....	456
Table 3-625. Function i2c_timing_config	457
Table 3-626. Function i2c_digital_noise_filter_config	457
Table 3-627. Function i2c_analog_noise_filter_enable	458
Table 3-628. Function i2c_analog_noise_filter_disable	459
Table 3-629. Function i2c_wakeup_from_deepsleep_enable.....	459
Table 3-630. Function i2c_wakeup_from_deepsleep_disable	460
Table 3-631. Function i2c_master_clock_config	460
Table 3-632. Function i2c2_master_transfer_direction_config.....	461
Table 3-633. Function i2c_address10_header_enable.....	462
Table 3-634. Function i2c_address10_header_disable.....	462
Table 3-635. Function i2c_address10_enable	463
Table 3-636. Function i2c_address10_disable	463
Table 3-637. Function i2c_automatic_end_enable	464
Table 3-638. Function i2c_automatic_end_disable	464
Table 3-639. Function i2c_address_config	465
Table 3-640. Function i2c_address_disable	465
Table 3-641. Function i2c_second_address_config.....	466
Table 3-642. Function i2c_second_address_disable	467
Table 3-643. Function i2c_recevied_address_get	467
Table 3-644. Function i2c_slave_byte_control_enable.....	468
Table 3-645. Function i2c_slave_byte_control_disable.....	468
Table 3-646. Function i2c_nack_enable	469
Table 3-647. Function i2c_nack_disable	469
Table 3-648. Function i2c_reload_enable.....	470
Table 3-649. Function i2c_reload_disable	470
Table 3-650. Function i2c_transfer_byte_number_config.....	471
Table 3-651. Function i2c2_dma_enable	471
Table 3-652. Function i2c2_dma_disable	472
Table 3-653. Function i2c_smbus_alert_enable.....	473
Table 3-654. Function i2c_smbus_alert_disable.....	473

Table 3-655. Function i2c_smbus_default_addr_enable	474
Table 3-656. Function i2c_smbus_default_addr_disable	474
Table 3-657. Function i2c_smbus_host_addr_enable	475
Table 3-658. Function i2c_smbus_host_addr_disable	475
Table 3-659. Function i2c_extented_clock_timeout_enable	476
Table 3-660. Function i2c_extented_clock_timeout_disable	476
Table 3-661. Function i2c_clock_timeout_enable	477
Table 3-662. Function i2c_clock_timeout_disable	477
Table 3-663. Function i2c_bus_timeout_b_config	478
Table 3-664. Function i2c_bus_timeout_a_config	478
Table 3-665. Function i2c_idle_clock_timeout_config	479
Table 3-666. Function i2c2_flag_get	479
Table 3-667. Function i2c2_flag_clear	480
Table 3-668. Function i2c2_interrupt_enable	481
Table 3-669. Function i2c2_interrupt_disable	482
Table 3-670. Function i2c2_interrupt_flag_get	483
Table 3-671. Function i2c2_interrupt_flag_clear	484
Table 3-672. NVIC Registers	485
Table 3-673. SysTick Registers	485
Table 3-674. MISC firmware function	486
Table 3-675. IRQn_Type	486
Table 3-676. Function nvic_priority_group_set	488
Table 3-677. Function nvic_irq_enable	489
Table 3-678. Function nvic_irq_disable	489
Table 3-679. Function nvic_system_reset	490
Table 3-680. Function nvic_vector_table_set	490
Table 3-681. Function system_lowpower_set	491
Table 3-682. Function system_lowpower_reset	492
Table 3-683. Function systick_clksource_set	492
Table 3-684. PMU Registers	493
Table 3-685. PMU firmware function	493
Table 3-686. Function pmu_deinit	494
Table 3-687. Function pmu_lvd_select	494
Table 3-688. Function pmu_lvd_disable	495
Table 3-689. Function pmu_highdriver_mode_enable	496
Table 3-690. Function pmu_highdriver_mode_disable	496
Table 3-691. Function pmu_highdriver_switch_select	496
Table 3-692. Function pmu_lowdriver_mode_enable	497
Table 3-693. Function pmu_lowdriver_mode_disable	498
Table 3-694. Function pmu_lowpower_driver_config	498
Table 3-695. Function pmu_normalpower_driver_config	499
Table 3-696. Function pmu_to_sleepmode	499
Table 3-697. Function pmu_to_deepsleepmode	500
Table 3-698. Function pmu_to_deepsleepmode_1	500

Table 3-699. Function pmu_to_deepsleepmode_2	501
Table 3-700. Function pmu_to_standbymode	502
Table 3-701. Function pmu_backup_write_enable	503
Table 3-702. Function pmu_backup_write_disable	503
Table 3-703. Function pmu_wakeup_pin_enable	504
Table 3-704. Function pmu_wakeup_pin_disable	504
Table 3-705. Function pmu_flag_clear	505
Table 3-706. Function pmu_flag_get	506
Table 3-707. RCU Registers	507
Table 3-708. RCU firmware function	507
Table 3-709. Enum rcu_periph_enum	509
Table 3-710. Enum rcu_periph_sleep_enum	510
Table 3-711. Enum rcu_periph_reset_enum	510
Table 3-712. Enum rcu_flag_enum	512
Table 3-713. Enum rcu_int_flag_enum	513
Table 3-714. Enum rcu_int_flag_clear_enum	513
Table 3-715. Enum rcu_int_enum	514
Table 3-716. Enum rcu_osc1_type_enum	514
Table 3-717. Enum rcu_clock_freq_enum	514
Table 3-718. Function rcu_deinit	515
Table 3-719. Function rcu_periph_clock_enable	515
Table 3-720. Function rcu_periph_clock_disable	516
Table 3-721. Function rcu_periph_clock_sleep_enable	516
Table 3-722. Function rcu_periph_clock_sleep_disable	517
Table 3-723. Function rcu_periph_reset_enable	517
Table 3-724. Function rcu_periph_reset_disable	518
Table 3-725. Function rcu_bkp_reset_enable	518
Table 3-726. Function rcu_bkp_reset_disable	519
Table 3-727. Function rcu_system_clock_source_config	519
Table 3-728. Function rcu_system_clock_source_get	520
Table 3-729. Function rcu_ahb_clock_config	520
Table 3-730. Function rcu_apb1_clock_config	521
Table 3-731. Function rcu_apb2_clock_config	521
Table 3-732. Function rcu_ckout0_config	522
Table 3-733. Function rcu_pll_config	523
Table 3-734. Function rcu_pllpresel_config	524
Table 3-735. Function rcu_pdevv0_config	524
Table 3-736. Function rcu_pdevv0_config	525
Table 3-737. Function rcu_pdevv1_config	526
Table 3-738. Function rcu_pll1_config	526
Table 3-739. Function rcu_pll2_config	527
Table 3-740. Function rcu_pllusbpresel_config	527
Table 3-741. Function rcu_pllusbpredv_config	528
Table 3-742. Function rcu_pllusb_config	528

Table 3-743. Function <code>rcu_adc_clock_config</code>	529
Table 3-744. Function <code>rcu_usb_clock_config</code>	530
Table 3-745. Function <code>rcu_rtc_clock_config</code>	531
Table 3-746. Function <code>rcu_shrtimer_clock_config</code>	531
Table 3-747. Function <code>rcu_usart5_clock_config</code>	532
Table 3-748. Function <code>rcu_i2c2_clock_config</code>	532
Table 3-749. Function <code>rcu_ck48m_clock_config</code>	533
Table 3-750. Function <code>rcu_i2s1_clock_config</code>	534
Table 3-751. Function <code>rcu_i2s2_clock_config</code>	534
Table 3-752. Function <code>rcu_usbhssel_config</code>	535
Table 3-753. Function <code>rcu_usbdv_config</code>	535
Table 3-754. Function <code>rcu_lxtal_drive_capability_config</code>	536
Table 3-755. Function <code>rcu_osc1_stab_wait</code>	537
Table 3-756. Function <code>rcu_osc1_on</code>	537
Table 3-757. Function <code>rcu_osc1_off</code>	538
Table 3-758. Function <code>rcu_osc1_bypass_mode_enable</code>	538
Table 3-759. Function <code>rcu_osc1_bypass_mode_disable</code>	539
Table 3-760. Function <code>rcu_irc8m_adjust_value_set</code>	539
Table 3-761. Function <code>rcu_hxtal_clock_monitor_enable</code>	540
Table 3-762. Function <code>rcu_hxtal_clock_monitor_disable</code>	540
Table 3-763. Function <code>rcu_deepsleep_voltage_set</code>	541
Table 3-764. Function <code>rcu_clock_freq_get</code>	541
Table 3-765. Function <code>rcu_flag_get</code>	542
Table 3-766. Function <code>rcu_all_reset_flag_clear</code>	543
Table 3-767. Function <code>rcu_interrupt_flag_get</code>	543
Table 3-768. Function <code>rcu_interrupt_flag_clear</code>	544
Table 3-769. Function <code>rcu_interrupt_enable</code>	544
Table 3-770. Function <code>rcu_interrupt_disable</code>	545
Table 3-771. RTC Registers	545
Table 3-772. RTC firmware function	546
Table 3-773. Function <code>rtc_configuration_mode_enter</code>	546
Table 3-774. Function <code>rtc_configuration_mode_exit</code>	547
Table 3-775. Function <code>rtc_lwoff_wait</code>	547
Table 3-776. Function <code>rtc_register_sync_wait</code>	548
Table 3-777. Function <code>rtc_counter_get</code>	548
Table 3-778. Function <code>rtc_counter_set</code>	549
Table 3-779. Function <code>rtc_prescaler_set</code>	549
Table 3-780. Function <code>rtc_alarm_config</code>	550
Table 3-781. Function <code>rtc_divider_get</code>	550
Table 3-782. Function <code>rtc_interrupt_enable</code>	551
Table 3-783. Function <code>rtc_interrupt_disable</code>	551
Table 3-784. Function <code>rtc_flag_get</code>	552
Table 3-785. Function <code>rtc_flag_clear</code>	553
Table 3-786. SDIO Registers	554

Table 3-787. SDIO firmware function	554
Table 3-788. Function <code>sdio_deinit</code>	555
Table 3-789. Function <code>sdio_clock_config</code>	556
Table 3-790. Function <code>sdio_hardware_clock_enable</code>	557
Table 3-791. Function <code>sdio_hardware_clock_disable</code>	557
Table 3-792. Function <code>sdio_bus_mode_set</code>	558
Table 3-793. Function <code>sdio_power_state_set</code>	558
Table 3-794. Function <code>sdio_power_state_get</code>	559
Table 3-795. Function <code>sdio_clock_enable</code>	559
Table 3-796. Function <code>sdio_clock_disable</code>	560
Table 3-797. Function <code>sdio_command_response_config</code>	560
Table 3-798. Function <code>sdio_wait_type_set</code>	561
Table 3-799. Function <code>sdio_csm_enable</code>	562
Table 3-800. Function <code>sdio_csm_disable</code>	562
Table 3-801. Function <code>sdio_command_index_get</code>	563
Table 3-802. Function <code>sdio_response_get</code>	563
Table 3-803. Function <code>sdio_data_config</code>	564
Table 3-804. Function <code>sdio_data_transfer_config</code>	565
Table 3-805. Function <code>sdio_dsm_enable</code>	566
Table 3-806. Function <code>sdio_dsm_disable</code>	566
Table 3-807. Function <code>sdio_data_write</code>	567
Table 3-808. Function <code>sdio_data_read</code>	567
Table 3-809. Function <code>sdio_data_counter_get</code>	568
Table 3-810. Function <code>sdio_data_counter_get</code>	568
Table 3-811. Function <code>sdio_dma_enable</code>	569
Table 3-812. Function <code>sdio_dma_disable</code>	569
Table 3-813. Function <code>sdio_flag_get</code>	570
Table 3-814. Function <code>sdio_flag_clear</code>	571
Table 3-815. Function <code>sdio_interrupt_enable</code>	572
Table 3-816. Function <code>sdio_interrupt_disable</code>	573
Table 3-817. Function <code>sdio_interrupt_flag_get</code>	574
Table 3-818. Function <code>sdio_interrupt_flag_clear</code>	576
Table 3-819. Function <code>sdio_readwait_enable</code>	577
Table 3-820. Function <code>sdio_readwait_disable</code>	577
Table 3-821. Function <code>sdio_stop_readwait_enable</code>	578
Table 3-822. Function <code>sdio_stop_readwait_disable</code>	578
Table 3-823. Function <code>sdio_readwait_type_set</code>	579
Table 3-824. Function <code>sdio_operation_enable</code>	579
Table 3-825. Function <code>sdio_operation_disable</code>	580
Table 3-826. Function <code>sdio_suspend_enable</code>	580
Table 3-827. Function <code>sdio_suspend_disable</code>	581
Table 3-828. Function <code>sdio_ceata_command_enable</code>	581
Table 3-829. Function <code>sdio_ceata_command_disable</code>	582
Table 3-830. Function <code>sdio_ceata_interrupt_enable</code>	582

Table 3-831. Function <code>sdio_ceata_interrupt_disable</code>	583
Table 3-832. Function <code>sdio_ceata_command_completion_enable</code>	583
Table 3-833. Function <code>sdio_ceata_command_completion_disable</code>	584
Table 3-834. SPI/I2S Registers	584
Table 3-835. SPI/I2S firmware function.....	585
Table 3-836. <code>spi_parameter_struct</code>	586
Table 3-837. Function <code>spi_i2s_deinit</code>	586
Table 3-838. Function <code>spi_struct_para_init</code>	587
Table 3-839. Function <code>spi_init</code>	587
Table 3-840. Function <code>spi_enable</code>	588
Table 3-841. Function <code>spi_disable</code>	589
Table 3-842. Function <code>i2s_init</code>	589
Table 3-843. Function <code>i2s_psc_config</code>	590
Table 3-844. Function <code>i2s_enable</code>	592
Table 3-845. Function <code>i2s_disable</code>	592
Table 3-846. Function <code>spi_nss_output_enable</code>	593
Table 3-847. Function <code>spi_nss_output_disable</code>	593
Table 3-848. Function <code>spi_nss_internal_high</code>	594
Table 3-849. Function <code>spi_nss_internal_low</code>	594
Table 3-850. Function <code>spi_dma_enable</code>	595
Table 3-851. Function <code>spi_dma_disable</code>	595
Table 3-852. Function <code>spi_i2s_data_frame_format_config</code>	596
Table 3-853. Function <code>spi_i2s_data_transmit</code>	596
Table 3-854. Function <code>spi_i2s_data_receive</code>	597
Table 3-855. Function <code>spi_bidirectional_transfer_config</code>	598
Table 3-856. Function <code>spi_i2s_format_error_clear</code>	598
Table 3-857. Function <code>spi_crc_polynomial_set</code>	599
Table 3-858. Function <code>spi_crc_polynomial_get</code>	599
Table 3-859. Function <code>spi_crc_on</code>	600
Table 3-860. Function <code>spi_crc_off</code>	600
Table 3-861. Function <code>spi_crc_next</code>	601
Table 3-862. Function <code>spi_crc_get</code>	601
Table 3-863. Function <code>spi_crc_error_clear</code>	602
Table 3-864. Function <code>spi_ti_mode_enable</code>	603
Table 3-865. Function <code>spi_ti_mode_disable</code>	603
Table 3-866. Function <code>spi_nssp_mode_enable</code>	604
Table 3-867. Function <code>spi_nssp_mode_disable</code>	604
Table 3-868. Function <code>i2s_init</code>	605
Table 3-869. Function <code>spi_quad_enable</code>	606
Table 3-870. Function <code>spi_quad_disable</code>	606
Table 3-871. Function <code>spi_quad_write_enable</code>	607
Table 3-872. Function <code>spi_quad_read_enable</code>	607
Table 3-873. Function <code>spi_quad_io23_output_enable</code>	608
Table 3-874. Function <code>spi_quad_io23_output_disable</code>	608

Table 3-875. Function spi_i2s_interrupt_enable	609
Table 3-876. Function spi_i2s_interrupt_disable	610
Table 3-877. Function spi_i2s_interrupt_flag_get	610
Table 3-878. Function spi_i2s_flag_get	611
Table 3-879. SQPI Registers	612
Table 3-880. SQPI firmware function	613
Table 3-881. sqpi_parameter_struct	613
Table 3-882. Function sqpi_deinit	613
Table 3-883. Function sqpi_struct_para_init	614
Table 3-884. Function sqpi_init	614
Table 3-885. Function sqpi_read_id_command	615
Table 3-886. Function sqpi_special_command	615
Table 3-887. Function sqpi_read_command_config	616
Table 3-888. Function sqpi_write_command_config	617
Table 3-889. Function sqpi_low_id_receive	617
Table 3-890. Function sqpi_low_id_receive	618
Table 3-891. TIMERx Registers	619
Table 3-892. TIMERx firmware function	619
Table 3-893. Structure timer_parameter_struct	622
Table 3-894. Structure timer_break_parameter_struct	622
Table 3-895. Structure timer_oc_parameter_struct	622
Table 3-896. Structure timer_ic_parameter_struct	623
Table 3-897. Function timer_deinit	623
Table 3-898. Function timer_struct_para_init	623
Table 3-899. Function timer_init	624
Table 3-900. Function timer_enable	625
Table 3-901. Function timer_disable	625
Table 3-902. Function timer_auto_reload_shadow_enable	626
Table 3-903. Function timer_auto_reload_shadow_disable	626
Table 3-904. Function timer_update_event_enable	627
Table 3-905. Function timer_update_event_disable	627
Table 3-906. Function timer_counter_alignment	628
Table 3-907. Function timer_counter_up_direction	629
Table 3-908. timer_counter_down_direction	629
Table 3-909. Function timer_prescaler_config	630
Table 3-910. Function timer_repetition_value_config	630
Table 3-911. Function timer_autoreload_value_config	631
Table 3-912. Function timer_counter_value_config	632
Table 3-913. Function timer_counter_read	632
Table 3-914. Function timer_prescaler_read	633
Table 3-915. Function timer_single_pulse_mode_config	633
Table 3-916. Function timer_update_source_config	634
Table 3-917. Function timer_dma_enable	635
Table 3-918. Function timer_dma_disable	635

Table 3-919. Function timer_channel_dma_request_source_select.....	636
Table 3-920. Function timer_dma_transfer_config.....	637
Table 3-921. Function timer_event_software_generate.....	638
Table 3-922. Function timer_break_struct_para_init	639
Table 3-923. Function timer_break_config	640
Table 3-924. Function timer_break_enable.....	641
Table 3-925. Function timer_break_disable.....	641
Table 3-926. Function timer_automatic_output_enable	642
Table 3-927. Function timer_automatic_output_disable	642
Table 3-928. Function timer_primary_output_config.....	643
Table 3-929. Function timer_channel_control_shadow_config	643
Table 3-930. Function timer_channel_control_shadow_update_config.....	644
Table 3-931. Function timer_channel_output_struct_para_init	645
Table 3-932. Function timer_channel_output_config	645
Table 3-933. Function timer_channel_output_mode_config	646
Table 3-934. Function timer_channel_output_pulse_value_config.....	647
Table 3-935. Function timer_channel_output_shadow_config	648
Table 3-936. Function timer_channel_output_fast_config.....	649
Table 3-937. Function timer_channel_output_clear_config	650
Table 3-938. Function timer_channel_output_polarity_config.....	651
Table 3-939. Function timer_channel_complementary_output_polarity_config	651
Table 3-940. Function timer_channel_output_state_config	652
Table 3-941. Function timer_channel_complementary_output_state_config	653
Table 3-942. Function timer_channel_input_struct_para_init	654
Table 3-943. Function timer_input_capture_config.....	654
Table 3-944. Function timer_channel_input_capture_prescaler_config	655
Table 3-945. Function timer_channel_capture_value_register_read	656
Table 3-946. Function timer_input_pwm_capture_config	657
Table 3-947. Function timer_hall_mode_config	658
Table 3-948. Function timer_input_trigger_source_select	658
Table 3-949. Function timer_master_output_trigger_source_select	659
Table 3-950. Function timer_slave_mode_select	660
Table 3-951. Function timer_master_slave_mode_config	661
Table 3-952. Function timer_external_trigger_config	662
Table 3-953. Function timer_quadrature_decoder_mode_config	663
Table 3-954. Function timer_internal_clock_config	664
Table 3-955. Function timer_internal_trigger_as_external_clock_config	665
Table 3-956. Function timer_external_trigger_as_external_clock_config	665
Table 3-957. Function timer_external_clock_mode0_config	666
Table 3-958. Function timer_external_clock_mode1_config	667
Table 3-959. Function timer_external_clock_mode1_disable	668
Table 3-960. Function timer_write_chxval_register_config	669
Table 3-961. Function timer_output_value_selection_config	669
Table 3-962. Function timer_flag_get	670

Table 3-963. Function timer_flag_clear	671
Table 3-964. Function timer_interrupt_enable	672
Table 3-965. Function timer_interrupt_disable	672
Table 3-966. Function timer_interrupt_flag_get.....	673
Table 3-967. Function timer_interrupt_flag_clear.....	674
Table 3-968. TMU Registers	675
Table 3-969. TMU firmware function	675
Table 3-970. Function tmu_deinit	676
Table 3-971. Function tmu_enable	676
Table 3-972. Function tmu_mode_set.....	677
Table 3-973. Function tmu_idata0_write	677
Table 3-974. Function tmu_idata1_write	678
Table 3-975. Function tmu_data0_read	678
Table 3-976. Function tmu_data1_read	679
Table 3-977. Function tmu_interrupt_enable.....	679
Table 3-978. Function tmu_interrupt_disable.....	680
Table 3-979. Function tmu_flag_get.....	680
Table 3-980. Function tmu_interrupt_flag_get	681
Table 3-981. USART Registers	681
Table 3-982. USART firmware function.....	682
Table 3-983. Enum usart_flag_enum.....	685
Table 3-984. Enum usart5_flag_enum	685
Table 3-985. Enum usart_interrupt_flag_enum.....	686
Table 3-986. Enum usart5_interrupt_flag_enum.....	686
Table 3-987. Enum usart_interrupt_enum.....	687
Table 3-988. Enum usart5_interrupt_enum.....	687
Table 3-989. Enum usart_invert_enum	687
Table 3-990. Enum usart5_invert_enum.....	688
Table 3-991. Function usart_deinit	688
Table 3-992. Function usart_baudrate_set	689
Table 3-993. Function usart_parity_config	689
Table 3-994. Function usart_word_length_set	690
Table 3-995. Function usart_stop_bit_set.....	690
Table 3-996. Function usart_enable	691
Table 3-997. Function usart_disable	692
Table 3-998. Function usart_transmit_config.....	692
Table 3-999. Function usart_receive_config	693
Table 3-1000. Function usart_oversample_config	693
Table 3-1001. Function usart_sample_bit_config.....	694
Table 3-1002. Function usart_receiver_timeout_enable	695
Table 3-1003. Function usart_receiver_timeout_disable	695
Table 3-1004. Function usart_receiver_timeout_threshold_config	696
Table 3-1005. Function usart_data_transmit	696
Table 3-1006. Function usart_data_receive	697

Table 3-1007. Function usart_mute_mode_enable.....	697
Table 3-1008. Function usart_mute_mode_disable	698
Table 3-1009. Function usart_mute_mode_wakeup_config	698
Table 3-1010. Function usart_lin_mode_enable	699
Table 3-1011. Function usart_lin_mode_disable	700
Table 3-1012. Function usart_lin_break_dection_length_config.....	700
Table 3-1013. Function usart_halfduplex_enable.....	701
Table 3-1014. Function usart_halfduplex_disable.....	701
Table 3-1015. Function usart_synchronous_clock_enable	702
Table 3-1016. Function usart_synchronous_clock_disable	702
Table 3-1017. Function usart_synchronous_clock_config.....	703
Table 3-1018. Function usart_guard_time_config	704
Table 3-1019. Function usart_smartcard_mode_enable	704
Table 3-1020. Function usart_smartcard_mode_disable	705
Table 3-1021. Function usart_smartcard_mode_nack_enable.....	705
Table 3-1022. Function usart_smartcard_mode_nack_disable	706
Table 3-1023. Function usart_smartcard_autoretry_config.....	706
Table 3-1024. Function usart_block_length_config	707
Table 3-1025. Function usart_irda_mode_enable.....	707
Table 3-1026. Function usart_irda_mode_disable.....	708
Table 3-1027. Function usart_prescaler_config.....	708
Table 3-1028. Function usart_irda_lowpower_config	709
Table 3-1029. Function usart_dma_receive_config	710
Table 3-1030. Function usart_dma_transmit_config.....	710
Table 3-1031. Function usart_hardware_flow_rts_config	711
Table 3-1032. Function usart_hardware_flow_cts_config	712
Table 3-1033. Function usart_data_first_config	712
Table 3-1034. Function usart_invert_config	713
Table 3-1035. Function usart_address_config	714
Table 3-1036. Function usart_send_break.....	714
Table 3-1037. Function usart_flag_get	715
Table 3-1038. Function usart_flag_clear	716
Table 3-1039. Function usart_interrupt_enable	716
Table 3-1040. Function usart_interrupt_disable	717
Table 3-1041. Function usart_interrupt_flag_get.....	718
Table 3-1042. Function usart_interrupt_flag_clear.....	719
Table 3-1043. Function usart5_data_first_config	720
Table 3-1044. Function usart5_invert_config	721
Table 3-1045. Function usart5_overrun_enable.....	722
Table 3-1046. Function usart5_overrun_disable.....	722
Table 3-1047. Function usart5_address_config.....	723
Table 3-1048. Function usart5_address_detection_mode_config.....	723
Table 3-1049. Function usart5_smartcard_mode_early_nack_enable	724
Table 3-1050. Function usart5_smartcard_mode_early_nack_disable	725

Table 3-1051. Function usart5_reception_error_dma_enable	725
Table 3-1052. Function usart5_reception_error_dma_disable	726
Table 3-1053. Function usart5_wakeup_enable	726
Table 3-1054. Function usart5_wakeup_disable	727
Table 3-1055. Function usart5_wakeup_mode_config	727
Table 3-1056. Function usart5_receive_fifo_enable	728
Table 3-1057. Function usart5_receive_fifo_disable	728
Table 3-1058. Function usart5_receive_fifo_counter_number	729
Table 3-1059. Function usart5_flag_get	729
Table 3-1060. Function usart5_flag_clear	730
Table 3-1061. Function usart5_interrupt_enable	731
Table 3-1062. Function usart5_interrupt_disable	732
Table 3-1063. Function usart5_command_enable	733
Table 3-1064. Function usart5_interrupt_flag_get	734
Table 3-1065. Function usart5_interrupt_flag_clear	735
Table 3-1066. WWDGT Registers	737
Table 3-1067. WWDGT firmware function	737
Table 3-1068. Function wwdgt_deinit	737
Table 3-1069. Function wwdgt_enable	738
Table 3-1070. Function wwdgt_counter_update	738
Table 3-1071. Function wwdgt_config	739
Table 3-1072. Function wwdgt_interrupt_enable	739
Table 3-1073. Function wwdgt_flag_get	740
Table 3-1074. Function wwdgt_flag_clear	741
Table 4-1. Revision history	742

1. Introduction

This manual introduces firmware library of GD32E50x 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 GD32E50x 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
CRC	CRC calculation unit
CTC	Clock trim controller

Peripherals	Descriptions
CMP	Comparator
DAC	Digital-to-analog converter
DBG	Debug
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
SHRTIMER	High-Precision Timer
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
SQPI	Serial/Quad Parallel Interface
TIMER	TIMER
TMU	Trigonometric Math Unit
USART	Universal synchronous/asynchronous receiver /transmitter
WWDGT	Window watchdog timer
USBD	Universal Serial Bus full-speed device interface
USBHS	Universal serial bus High-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 “gd32e50x_”, such as: gd32e50x_adc.h;
- The constants used only in one file should be defined in the used file; the constants used in many files should be defined in corresponding header file. All the constants are written in uppercase of English letters;
- Registers are handled as constants. The naming of them are written in uppercase of English letters. In most cases, register names are shortened accord with the user manual;
- Variables are written in lowercase, when concluded by several words, underlines should

be adapted among words;

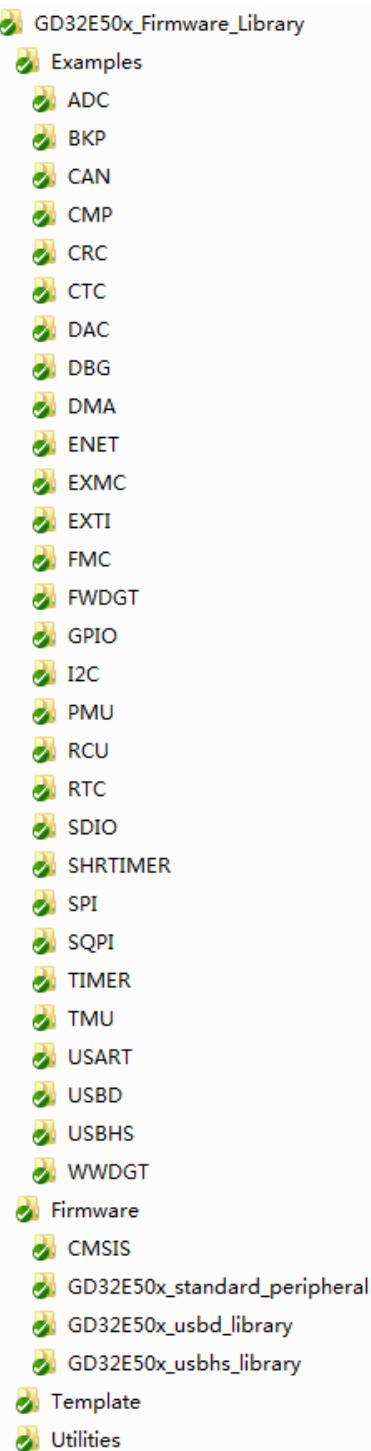
- The naming of peripheral functions are started with the peripheral abbreviation added with an underline, when the function name is concluded by several words, underlines should be adapted among words, and all the peripheral functions are written in lowercase.

2. Firmware Library Overview

2.1. File Structure of Firmware Library

GD32E50x_Firmware_Library, the file structure is shown as below:

Figure 2-1. File structure of firmware library of GD32E50x



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;
- gd32e50x_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);
- gd32e50x_it.c: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- gd32e50x_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 M33 kernel support files, the startup file based on the Cortex M33 kernel processor, the global header file of GD32E50x and system configuration file;
- GD32E50x_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;
- GD32E50x_usbd_library subfolder includes all the related files about USBD peripheral:
 - Include subfolder includes the header files of USBD peripheral, users need not modify this folder;
 - Source subfolder includes the source files of USBD peripheral, users need not modify this folder;
- GD32E50x_usbhs_library subfolder includes all the related files about USBHS peripheral:
 - Include subfolder includes the header files of USBHS peripheral, users need not modify this folder;
 - Source subfolder includes the source files of USBHS 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.

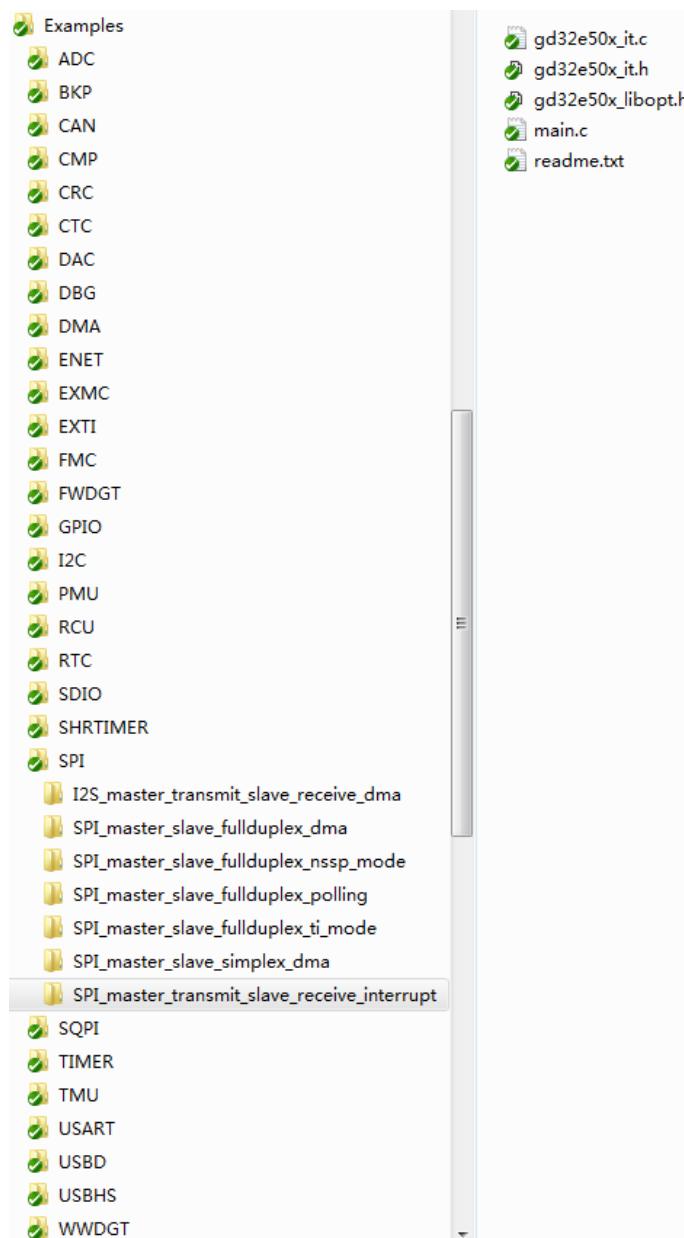
2.1.3. Template Folder

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 Keil5). User can use the project template to compile the firmware examples, the steps are shown as below:

Select files

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

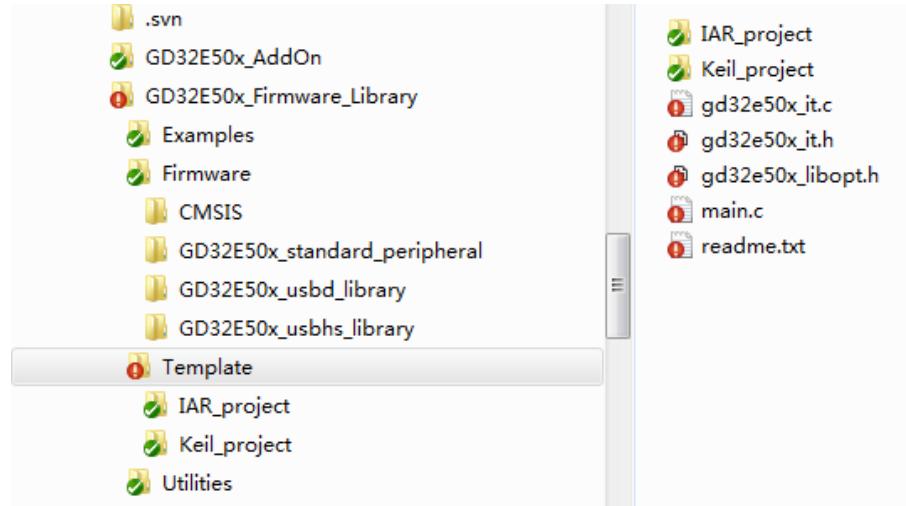
Figure 2-2. Select peripheral example files



Copy files

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

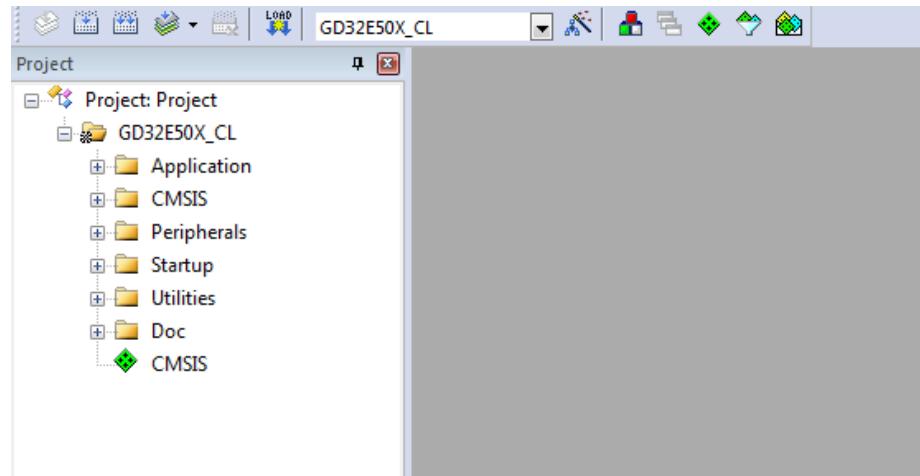
Figure 2-3. Copy the peripheral example files



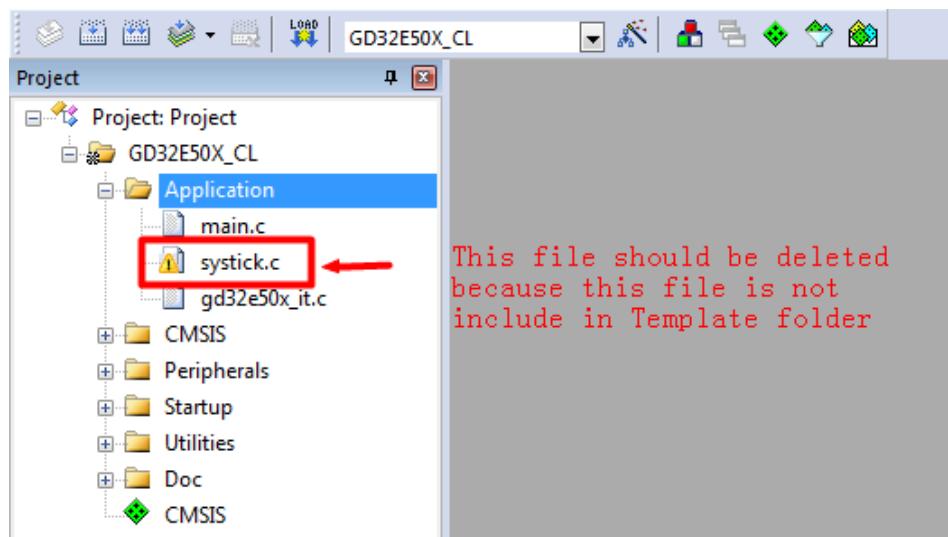
Open a project

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

Figure 2-4. Open the project file

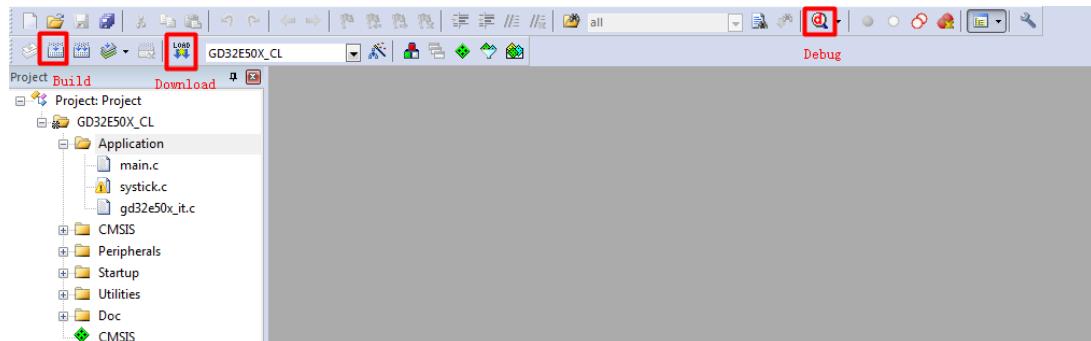


Because different module and different functions adopt different files, users should add or delete the files in project according to the copied files, shown as below:

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_Common subfolders include files for USB tests;
- gd32e50x_eval.h and gd32e50x_lcd_eval.h are related header files of the evaluation board about running the firmware examples;
- gd32e50x_eval.c and gd32e50x_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
gd32e50x_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.
gd32e50x_it.h	Header file, including all the prototypes of interrupt service routines.
gd32e50x_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.
gd32e50x_xxx.h	The header file of peripheral PPP, including functions about peripheral PPP, and the variables used for functions.
gd32e50x_xxx.c	The C source file for driving peripheral PPP.
systick.h	The header file of systick.c, including prototypes of systick configuration function and delay function.
systick.c	The source file about systick configuration function and delay function.
readme.txt	Description document about how to configure and how to use the firmware example.

3. Firmware Library of Standard Peripherals

3.1. Overview of Firmware Library of Standard Peripherals

The description format of firmware functions are shown as below:

Table 3-1. Peripheral function format of Firmware Library

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

3.2. ADC

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

3.2.1. Descriptions of Peripheral registers

ADC registers are listed in the table shown as below:

Table 3-2. ADC Registers

Registers	Descriptions
ADC_STAT	Status register
ADC_CTL0	Control register 0
ADC_CTL1	Control register 1
ADC_SAMPT0	Sample time register 0
ADC_SAMPT1	Sample time register 1
ADC_IOFFx (x=0..3)	Inserted channel data offset register x(x=0..3)
ADC_WDHT0	Watchdog 0 high threshold register

Registers	Descriptions
ADC_WDLT0	Watchdog 0 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 (x=0..3)	Inserted data register x(x=0..3)
ADC_RDATA	Regular data register
ADC_OVSAMPCTL	Oversample control register
ADC_WD1SR	ADC watchdog 1 channel selection register
ADC_WD2SR	ADC watchdog 2 channel selection register
ADC_WDT1	ADC watchdog threshold register 1
ADC_WDT2	ADC watchdog threshold register 2
ADC_DIFCTL	ADC differential mode 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 ADC peripheral
adc_enable	enable ADC interface
adc_disable	disable ADC interface
adc_calibration_enable	ADC calibration and reset calibration
adc_calibration_number	configure ADC calibration number
adc_dma_mode_enable	enable DMA request
adc_dma_mode_disable	disable DMA request
adc_tempsensor_vrefint_enable	enable the temperature sensor and vrefint channel
adc_tempsensor_vrefint_disable	disable the temperature sensor and vrefint channel
adc_discontinuous_mode_config	configure ADC discontinuous mode
adc_mode_config	configure the ADC sync mode
adc_special_function_config	enable or disable ADC special function
adc_data_alignment_config	configure ADCx data alignment
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_channel_differential_mode_config	configure differential mode for ADC channel
adc_external_trigger_config	enable ADC external trigger

Function name	Function description
adc_external_trigger_source_config	configure ADC external trigger source
adc_software_trigger_enable	enable ADC software trigger
adc_regular_data_read	read ADC regular group data register
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_watchdog0_single_channel_enable	configure ADC analog watchdog 0 single channel
adc_watchdog0_group_channel_enable	configure ADC analog watchdog 0 group channel
adc_watchdog0_disable	disable ADC analog watchdog 0
adc_watchdog1_channel_config	configure ADC analog watchdog 1 channel
adc_watchdog2_channel_config	configure ADC analog watchdog 2 channel
adc_watchdog1_disable	disable ADC analog watchdog 1
adc_watchdog2_disable	disable ADC analog watchdog 2
adc_watchdog0_threshold_config	configure ADC analog watchdog 0 threshold
adc_watchdog1_threshold_config	configure ADC analog watchdog 1 threshold
adc_watchdog2_threshold_config	configure ADC analog watchdog 2 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 ADC peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset ADC0 */

adc_deinit(ADC0);
```

adc_enable

The description of adc_enable is shown as below:

Table 3-5. Function adc_enable

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

Example:

```
/* enable ADC0 */

adc_enable(ADC0);
```

adc_disable

The description of adc_disable is shown as below:

Table 3-6. Function adc_disable

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

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

adc_calibration_enable

The description of adc_calibration_enable is shown as below:

Table 3-7. Function adc_calibration_enable

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

Example:

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

adc_calibration_number

The description of adc_calibration_number is shown as below:

Table 3-8. Function adc_calibration_number

Function name	adc_calibration_number
Function prototype	void adc_calibration_number(uint32_t adc_periph, uint32_t clb_num);
Function descriptions	configure ADC calibration number
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection

Input parameter{in}	
clb_num	calibration number
<i>ADC_CALIBRATION_NUM1</i>	calibrate once
<i>ADC_CALIBRATION_NUM2</i>	calibrate twice
<i>ADC_CALIBRATION_NUM4</i>	calibrate 4 times
<i>ADC_CALIBRATION_NUM8</i>	calibrate 8 times
<i>ADC_CALIBRATION_NUM16</i>	calibrate 16 times
<i>ADC_CALIBRATION_NUM32</i>	calibrate 32 times
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC calibration number */

adc_calibration_number(ADC0, ADC_CALIBRATION_NUM16);
```

adc_dma_mode_enable

The description of `adc_dma_mode_enable` is shown as below:

Table 3-9. 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 ADC DMA request
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC0 DMA request */
```

```
adc_dma_mode_enable(ADC0);
```

adc_dma_mode_disable

The description of adc_dma_mode_disable is shown as below:

Table 3-10. 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 ADC DMA request
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 DMA request */
adc_dma_mode_disable(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_discontinuous_mode_config

The description of adc_discontinuous_mode_config is shown as below:

Table 3-13. Function adc_discontinuous_mode_config

Function name	adc_discontinuous_mode_config
Function prototype	void adc_discontinuous_mode_config(uint32_t adc_periph, uint8_t adc_channel_group, uint8_t length);
Function descriptions	configure ADC discontinuous mode
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_channel_group	select the channel group
ADC_REGULAR_CHANNEL	regular channel group
ADC_INSERTED_CHANNEL	inserted channel group
ADC_CHANNEL_DISCON_DISABLE	disable discontinuous mode of regular and inserted channel
Input parameter{in}	

length	number of conversions in discontinuous mode, the number can be 1..8 for regular channel, the number has no effect for inserted channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

adc_mode_config

The description of adc_mode_config is shown as below:

Table 3-14. 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
<i>ADC_MODE_FREE</i>	all the ADCs work independently
<i>ADC_DAUL_REGULAR_PARALLEL_INSERTED_D_PARALLEL</i>	ADC0 and ADC1 work in combined regular parallel + inserted parallel mode
<i>ADC_DAUL_REGULAR_PARALLEL_INSERTED_D_ROTATION</i>	ADC0 and ADC1 work in combined regular parallel + trigger rotation mode
<i>ADC_DAUL_INSERTED_D_PARALLEL_REGULAR_FOLLOWUP_FAST</i>	ADC0 and ADC1 work in combined inserted parallel + follow-up fast mode
<i>ADC_DAUL_INSERTED_D_PARALLEL_REGULAR_FOLLOWUP_SLOW</i>	ADC0 and ADC1 work in combined inserted parallel + follow-up slow mode
<i>ADC_DAUL_INSERTED_D_PARALLEL</i>	ADC0 and ADC1 work in inserted parallel mode only
<i>ADC_DAUL_REGULAR_PARALLEL</i>	ADC0 and ADC1 work in regular parallel mode only
<i>ADC_DAUL_REGULAR_FOLLOWUP_FAST</i>	ADC0 and ADC1 work in follow-up fast mode only

<i>ADC_DAUL_REGULAR_FOLLOWUP_SLOW</i>	ADC0 and ADC1 work in follow-up slow mode only
<i>ADC_DAUL_INSERTED_TRRIGGER_ROTATION</i>	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 3-15. 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 newvalue);</code>
Function descriptions	enable or disable ADC special function
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
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 3-16. Function adc_data_alignment_config

Function name	adc_data_alignment_config
Function prototype	void adc_data_alignment_config(uint32_t adc_periph, uint32_t data_alignment);
Function descriptions	configure ADCx data alignment
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
data_alignment	data alignment select
ADC_DATAALIGN_RIGHT	right alignment
ADC_DATAALIGN_LEFT	left alignment
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 data alignment */

adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);
```

adc_channel_length_config

The description of adc_channel_length_config is shown as below:

Table 3-17. 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(x=0..2)</i>		ADC peripheral selection
Input parameter{in}		
adc_channel_group		select the channel group
<i>ADC_REGULAR_CHANNEL</i>		regular channel group
<i>ADC_INSERTED_CHANNEL</i>		inserted channel group
Input parameter{in}		
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 3-18. 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(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
rank	the regular group sequence rank, this parameter must be between 0 to 15
Input parameter{in}	
adc_channel	the selected ADC channel
<i>ADC_CHANNEL_x (x=0..17)</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_1POINT5</i>	1.5 cycles
<i>ADC_SAMPLETIME_7POINT5</i>	7.5 cycles
<i>ADC_SAMPLETIME_13POINT5</i>	13.5 cycles
<i>ADC_SAMPLETIME_28POINT5</i>	28.5 cycles
<i>ADC_SAMPLETIME_41POINT5</i>	41.5 cycles
<i>ADC_SAMPLETIME_55POINT5</i>	55.5 cycles
<i>ADC_SAMPLETIME_71POINT5</i>	71.5 cycles
<i>ADC_SAMPLETIME_239POINT5</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 3-19. Function `adc_inserted_channel_config`

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

ADC_CHANNEL_X (x=0..17)	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_1POINT5	1.5 cycles
ADC_SAMPLETIME_7POINT5	7.5 cycles
ADC_SAMPLETIME_13POINT5	13.5 cycles
ADC_SAMPLETIME_28POINT5	28.5 cycles
ADC_SAMPLETIME_41POINT5	41.5 cycles
ADC_SAMPLETIME_55POINT5	55.5 cycles
ADC_SAMPLETIME_71POINT5	71.5 cycles
ADC_SAMPLETIME_239POINT5	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 3-20. 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..2)	ADC peripheral selection

Input parameter{in}	
inserted_channel	insert channel select
ADC_INSERTED_CHANNEL_x(x=0..3)	inserted channel, x=0,1,2,3
Input parameter{in}	
offset	the offset data, this parameter must be between 0 to 4095
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

adc_inserted_channel_offset_config(ADC0, ADC_INSERTED_CHANNEL_0, 100);
```

adc_channel_differential_mode_config

The description of `adc_channel_differential_mode_config` is shown as below:

Table 3-21. Function `adc_channel_differential_mode_config`

Function name	adc_channel_differential_mode_config
Function prototype	void adc_channel_differential_mode_config(uint32_t adc_periph, uint32_t adc_channel, ControlStatus newvalue);
Function descriptions	configure differential mode for ADC channel
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
inserted_channel	the channel use differential mode
ADC_DIFFERENTIAL_MODE_CHANNEL_x (<i>x</i> =0..14), ADC_DIFFERENTIAL_MODE_CHANNEL_ALL	ADC channel for differential mode (just for channel0~channel14)
Input parameter{in}	
newvalue	control value
ENABLE	enable function
DISABLE	disable function
Output parameter{out}	
-	-
Return value	

Example:

```
/* configure differential mode for ADC channel */

adc_channel_differential_mode_config(ADC0,
ADC_DIFFERENTIAL_MODE_CHANNEL_ALL, ENABLE);
```

adc_external_trigger_config

The description of adc_external_trigger_config is shown as below:

Table 3-22. Function adc_external_trigger_config

Function name	adc_external_trigger_config
Function prototype	void adc_external_trigger_config(uint32_t adc_periph, uint8_t adc_channel_group, ControlStatus newvalue);
Function descriptions	configure ADC external trigger
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_channel_group	select the channel group
ADC_REGULAR_CHANNEL	regular channel group
ADC_INSERTED_CHANNEL	inserted channel group
Input parameter{in}	
newvalue	control value
ENABLE	enable function
DISABLE	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_external_trigger_source_config

The description of adc_external_trigger_source_config is shown as below:

Table 3-23. 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}	
<code>adc_periph</code>	ADC peripheral
<code>ADCx(x=0..2)</code>	ADC peripheral selection
Input parameter{in}	
<code>adc_channel_group</code>	select the channel group
<code>ADC_REGULAR_CHANNEL</code>	regular channel group
<code>ADC_INSERTED_CHANNEL</code>	inserted channel group
Input parameter{in}	
<code>external_trigger_source</code>	regular or inserted group trigger source
<code>ADC0_1_EXTTRIG_REGULAR_T0_CH0</code>	TIMER0 CH0 event select for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_T0_CH1</code>	TIMER0 CH1 event select for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_T0_CH2</code>	TIMER0 CH2 event select for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_T1_CH1</code>	TIMER1 CH1 event select for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_T2_TRGO</code>	TIMER2 TRGO event select for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_T3_CH3</code>	TIMER3 CH3 event select for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_T7_TRGO</code>	TIMER7 TRGO event select for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_EXTI_11</code>	external interrupt line 11 for regular channel
<code>ADC0_1_EXTTRIG_REGULAR_SHRTIMER_ADCTRG0</code>	SHRTIMER_ADCTRG0 output select for regular channel (for GD32E50X_HD and GD32E50X_CL devices)
<code>ADC0_1_EXTTRIG_REGULAR_SHRTIMER_ADCTRG2</code>	SHRTIMER_ADCTRG2 output select for regular channel (for GD32E50X_HD and GD32E50X_CL devices)
<code>ADC2_EXTTRIG_REGULAR_T2_CH0</code>	TIMER2 CH0 event select for regular channel

<i>ADC2_EXTTRIG_</i> <i>REGULAR_T1_CH2</i>	TIMER1 CH2 event select for regular channel
<i>ADC2_EXTTRIG_</i> <i>REGULAR_T0_CH2</i>	TIMER0 CH2 event select for regular channel
<i>ADC2_EXTTRIG_</i> <i>REGULAR_T7_CH0</i>	TIMER7 CH0 event select for regular channel
<i>ADC2_EXTTRIG_</i> <i>REGULAR_T7_TRGO</i>	TIMER7 TRGO event select for regular channel
<i>ADC2_EXTTRIG_</i> <i>REGULAR_T4_CH0</i>	TIMER4 CH0 event select for regular channel
<i>ADC2_EXTTRIG_</i> <i>REGULAR_T4_CH2</i>	TIMER4 CH2 event select for regular channel
<i>ADC0_1_2_EXTTRIG_</i> <i>REGULAR_NONE</i>	software trigger for regular channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_T0_TRGO</i>	TIMER0 TRGO event select for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_T0_CH3</i>	TIMER0 CH3 event select for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_T1_TRGO</i>	TIMER1 TRGO event select for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_T1_CH0</i>	TIMER1 CH0 event select for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_T2_CH3</i>	TIMER2 CH3 event select for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_T3_TRGO</i>	TIMER3 TRGO event select for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_T7_CH3</i>	TIMER7 CH3 event select for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_EXTI_15</i>	external interrupt line 15 for inserted channel
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_SHRTIME</i> <i>R_ADCTRG1</i>	SHRTIMER_ADCTRG1 output select for inserted channel (for GD32E50X_HD and GD32E50X_CL devices)
<i>ADC0_1_EXTTRIG_</i> <i>INSERTED_SHRTIME</i> <i>R_ADCTRG3</i>	SHRTIMER_ADCTRG3 output select for inserted channel (for GD32E50X_HD and GD32E50X_CL devices)
<i>ADC2_EXTTRIG_</i> <i>INSERTED_T0_TRGO</i>	TIMER0 TRGO event select for inserted channel
<i>ADC2_EXTTRIG_</i> <i>INSERTED_T0_CH3</i>	TIMER0 CH3 event select for inserted channel
<i>ADC2_EXTTRIG_</i> <i>INSERTED_T3_CH2</i>	TIMER3 CH2 event select for inserted channel
<i>ADC2_EXTTRIG_</i>	TIMER7 CH1 event select for inserted channel

<i>INSERTED_T7_CH1</i>	
<i>ADC2_EXTTRIG_INSERTED_T7_CH3</i>	TIMER7 CH3 event select for inserted channel
<i>ADC2_EXTTRIG_INSERTED_T4_TRGO</i>	TIMER4 TRGO event select for inserted channel
<i>ADC2_EXTTRIG_INSERTED_T4_CH3</i>	TIMER4 CH3 event select for inserted channel
<i>ADC0_1_2_EXTTRIG_INSERTED_NONE</i>	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_software_trigger_enable

The description of `adc_software_trigger_enable` is shown as below:

Table 3-24. 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}	
<code>adc_periph</code>	ADC peripheral
<code>ADCx(x=0..2)</code>	ADC peripheral selection
Input parameter{in}	
<code>adc_channel_group</code>	select the channel group
<code>ADC_REGULAR_CHANNEL</code>	regular channel group
<code>ADC_INSERTED_CHANNEL</code>	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 3-25. 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
ADCx(x=0..2)	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
uint16_t	ADC conversion value (0~0xFFFF)

Example:

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

uint16_t adc_value = 0;

adc_value = adc_regular_data_read(ADC0);
```

adc_inserted_data_read

The description of adc_inserted_data_read is shown as below:

Table 3-26. 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
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
inserted_channel	insert channel select

ADC_INSERTED_CHANNEL_x(x=0..3)	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 3-27. 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_watchdog0_single_channel_enable

The description of adc_watchdog0_single_channel_enable is shown as below:

Table 3-28. Function adc_watchdog0_single_channel_enable

Function name	adc_watchdog0_single_channel_enable
Function prototype	void adc_watchdog0_single_channel_enable(uint32_t adc_periph, uint8_t adc_channel);
Function descriptions	configure ADC analog watchdog 0 single channel

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

Example:

```
/* configure ADC0 analog watchdog 0 single channel */
adc_watchdog0_single_channel_enable(ADC0, ADC_CHANNEL_1);
```

adc_watchdog0_group_channel_enable

The description of `adc_watchdog0_group_channel_enable` is shown as below:

Table 3-29. Function `adc_watchdog0_group_channel_enable`

Function name	adc_watchdog0_group_channel_enable
Function prototype	void adc_watchdog0_group_channel_enable(uint32_t adc_periph, uint8_t adc_channel_group);
Function descriptions	configure ADC analog watchdog 0 group channel
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
adc_channel_group	the channel group use analog watchdog
<i>ADC_REGULAR_CHANNEL</i>	regular channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
<i>ADC_REGULAR_INSERTED_CHANNEL</i>	both regular and inserted group
Output parameter{out}	
-	-
Return value	

Example:

```
/* configure ADC0 analog watchdog 0 group channel */

adc_watchdog0_group_channel_enable(ADC0, ADC_REGULAR_CHANNEL);
```

adc_watchdog0_disable

The description of adc_watchdog0_disable is shown as below:

Table 3-30. Function adc_watchdog0_disable

Function name	adc_watchdog0_disable
Function prototype	void adc_watchdog0_disable(uint32_t adc_periph);
Function descriptions	disable ADC analog watchdog 0
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 analog watchdog 0 */

adc_watchdog0_disable(ADC0);
```

adc_watchdog1_channel_config

The description of adc_watchdog1_channel_config is shown as below:

Table 3-31. Function adc_watchdog1_channel_config

Function name	adc_watchdog1_channel_config
Function prototype	void adc_watchdog1_channel_config(uint32_t adc_periph, uint32_t adc_channel, ControlStatus newvalue);
Function descriptions	configure ADC analog watchdog 1 channel
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_channel	the selected ADC channel

<i>ADC_AWD1_2_SELECTION_CHANNEL_X</i> (<i>x</i> =0..17), <i>ADC_AWD1_2_SELECTION_CHANNEL_ALL</i>	ADC channel analog watchdog 1/2 selection (<i>x</i> =0..17, <i>x</i> =16 and <i>x</i> =17 are only for ADC0)
Input parameter{in}	
<i>newvalue</i>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC analog watchdog 1 channel */

adc_watchdog1_channel_config(ADC0, ADC_AWD1_2_SELECTION_CHANNEL_1,
ENABLE);
```

adc_watchdog2_channel_config

The description of `adc_watchdog2_channel_config` is shown as below:

Table 3-32. Function `adc_watchdog2_channel_config`

Function name	adc_watchdog2_channel_config
Function prototype	void adc_watchdog2_channel_config(uint32_t adc_periph, uint32_t adc_channel, ControlStatus newvalue);
Function descriptions	configure ADC analog watchdog 2 channel
Precondition	-
The called functions	-
Input parameter{in}	
<i>adc_periph</i>	ADC peripheral
<i>ADCx</i> (<i>x</i> =0..2)	ADC peripheral selection
Input parameter{in}	
<i>adc_channel</i>	the selected ADC channel
<i>ADC_AWD1_2_SELECTION_CHANNEL_X</i> (<i>x</i> =0..17), <i>ADC_AWD1_2_SELECTION_CHANNEL_ALL</i>	ADC channel analog watchdog 1/2 selection (<i>x</i> =0..17, <i>x</i> =16 and <i>x</i> =17 are only for ADC0)
Input parameter{in}	
<i>newvalue</i>	control value
<i>ENABLE</i>	enable function

DISABLE	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC analog watchdog 2 channel */

adc_watchdog2_channel_config(ADC0, ADC_AWD1_2_SELECTION_CHANNEL_1,
ENABLE);
```

adc_watchdog1_disable

The description of `adc_watchdog1_disable` is shown as below:

Table 3-33. Function `adc_watchdog1_disable`

Function name	adc_watchdog1_disable
Function prototype	void adc_watchdog1_disable(uint32_t adc_periph);
Function descriptions	disable ADC analog watchdog 1
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 analog watchdog 1 */

adc_watchdog1_disable(ADC0);
```

adc_watchdog2_disable

The description of `adc_watchdog2_disable` is shown as below:

Table 3-34. Function `adc_watchdog2_disable`

Function name	adc_watchdog2_disable
Function prototype	void adc_watchdog2_disable(uint32_t adc_periph);
Function descriptions	disable ADC analog watchdog 2
Precondition	-
The called functions	-
Input parameter{in}	

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

Example:

```
/* disable ADC0 analog watchdog 2 */
adc_watchdog2_disable(ADC0);
```

adc_watchdog0_threshold_config

The description of `adc_watchdog0_threshold_config` is shown as below:

Table 3-35. Function `adc_watchdog0_threshold_config`

Function name	adc_watchdog0_threshold_config
Function prototype	void adc_watchdog0_threshold_config(uint32_t adc_periph , uint16_t low_threshold , uint16_t high_threshold);
Function descriptions	configure ADC analog watchdog 0 threshold
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
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 0 threshold */
adc_watchdog0_threshold_config(ADC0, 0x0400, 0x0A00);
```

adc_watchdog1_threshold_config

The description of `adc_watchdog1_threshold_config` is shown as below:

Table 3-36. Function `adc_watchdog1_threshold_config`

Function name	adc_watchdog1_threshold_config
----------------------	--------------------------------

Function prototype	void adc_watchdog1_threshold_config(uint32_t adc_periph , uint8_t low_threshold , uint8_t high_threshold);
Function descriptions	configure ADC analog watchdog 1 threshold
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
low_threshold	analog watchdog low threshold, 0..255
Input parameter{in}	
high_threshold	analog watchdog high threshold, 0..255
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 analog watchdog 1 threshold */
adc_watchdog1_threshold_config(ADC0, 0x40, 0xA0);
```

adc_watchdog2_threshold_config

The description of adc_watchdog2_threshold_config is shown as below:

Table 3-37. Function adc_watchdog2_threshold_config

Function name	adc_watchdog2_threshold_config
Function prototype	void adc_watchdog2_threshold_config(uint32_t adc_periph , uint8_t low_threshold , uint8_t high_threshold);
Function descriptions	configure ADC analog watchdog 2 threshold
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
low_threshold	analog watchdog low threshold, 0..255
Input parameter{in}	
high_threshold	analog watchdog high threshold, 0..255
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 analog watchdog 2 threshold */

adc_watchdog2_threshold_config(ADC0, 0x40, 0xA0);
```

adc_resolution_config

The description of adc_resolution_config is shown as below:

Table 3-38. Function adc_resolution_config

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

Example:

```
/* configure ADC0 data alignment */

adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);
```

adc_oversample_mode_config

The description of adc_oversample_mode_config is shown as below:

Table 3-39. Function adc_oversample_mode_config

Function name	adc_oversample_mode_config
Function prototype	void adc_oversample_mode_config(uint32_t adc_periph, uint32_t mode,

	uint16_t shift, uint8_t ratio);
Function descriptions	configure ADC oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
mode	ADC oversampling mode
ADC_OVERSAMPLING_ALL_CONVERT	all oversampled conversions for a channel are done consecutively after a trigger
ADC_OVERSAMPLING_ONE_CONVERT	each oversampled conversion for a channel needs a trigger
Input parameter{in}	
shift	ADC oversampling shift
ADC_OVERSAMPLING_SHIFT_NONE	no oversampling shift
ADC_OVERSAMPLING_SHIFT_1B	1-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_2B	2-bit oversampling shift
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	oversampling ratio multiple 16

<code>_RATIO_MUL16</code>	
<code>ADC_OVERSAMPLING</code>	oversampling ratio multiple 32
<code>_RATIO_MUL32</code>	
<code>ADC_OVERSAMPLING</code>	oversampling ratio multiple 64
<code>_RATIO_MUL64</code>	
<code>ADC_OVERSAMPLING</code>	oversampling ratio multiple 128
<code>_RATIO_MUL128</code>	
<code>ADC_OVERSAMPLING</code>	oversampling ratio multiple 256
<code>_RATIO_MUL256</code>	
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

adc_oversample_mode_enable

The description of `adc_oversample_mode_enable` is shown as below:

Table 3-40. Function `adc_oversample_mode_enable`

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

Example:

```
/* enable ADC0 oversample mode */

adc_oversample_mode_enable (ADC0);
```

adc_oversample_mode_disable

The description of adc_oversample_mode_disable is shown as below:

Table 3-41. Function adc_oversample_mode_disable

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

Example:

```
/* disable ADC0 oversample mode */

adc_oversample_mode_disable (ADC0);
```

adc_flag_get

The description of adc_flag_get is shown as below:

Table 3-42. Function adc_flag_get

Function name	adc_flag_get
Function prototype	FlagStatus adc_flag_get(uint32_t adc_periph, uint32_t flag);
Function descriptions	get the ADC flag
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
flag	the adc flag bits
<i>ADC_FLAG_WDE0</i>	analog watchdog 0 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
<i>ADC_FLAG_WDE1</i>	analog watchdog 1 event flag
<i>ADC_FLAG_WDE2</i>	analog watchdog 2 event flag

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

Example:

```
/* get the ADC0 analog watchdog 0 flag bits */

FlagStatus flag_value;

flag_value = adc_flag_get(ADC0, ADC_FLAG_WDE0);
```

adc_flag_clear

The description of adc_flag_clear is shown as below:

Table 3-43. Function adc_flag_clear

Function name	adc_flag_clear
Function prototype	void adc_flag_clear(uint32_t adc_periph, uint32_t flag);
Function descriptions	clear the ADC flag bits
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
flag	the adc flag bits
ADC_FLAG_WDE0	analog watchdog 0 event flag
ADC_FLAG_EOC	end of group conversion flag
ADC_FLAG_EOIC	end of inserted group conversion flag
ADC_FLAG_STIC	start flag of inserted channel group
ADC_FLAG_STRC	start flag of regular channel group
ADC_FLAG_WDE1	analog watchdog 1 event flag
ADC_FLAG_WDE2	analog watchdog 2 event flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the ADC0 analog watchdog 0 flag bits */

adc_flag_clear(ADC0, ADC_FLAG_WDE0);
```

adc_interrupt_enable

The description of adc_interrupt_enable is shown as below:

Table 3-44. Function adc_interrupt_enable

Function name	adc_interrupt_enable
Function prototype	void adc_interrupt_enable(uint32_t adc_periph, uint32_t interrupt);
Function descriptions	enable ADC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
interrupt	the adc interrupt
ADC_INT_WDE0	analog watchdog 0 interrupt
ADC_INT_EOC	end of group conversion interrupt
ADC_INT_EOIC	end of inserted group conversion interrupt
ADC_INT_WDE1	analog watchdog 1 interrupt
ADC_INT_WDE2	analog watchdog 2 interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC0 analog watchdog 0 interrupt */
adc_interrupt_enable(ADC0, ADC_INT_WDE0);
```

adc_interrupt_disable

The description of adc_interrupt_disable is shown as below:

Table 3-45. Function adc_interrupt_disable

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

<i>ADC_INT_WDE0</i>	analog watchdog 0 interrupt
<i>ADC_INT_EOC</i>	end of group conversion interrupt
<i>ADC_INT_EOIC</i>	end of inserted group conversion interrupt
<i>ADC_INT_WDE1</i>	analog watchdog 1 interrupt
<i>ADC_INT_WDE2</i>	analog watchdog 2 interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 analog watchdog 0 interrupt */
adc_interrupt_disable(ADC0, ADC_INT_WDE0);
```

adc_interrupt_flag_get

The description of `adc_interrupt_flag_get` is shown as below:

Table 3-46. Function `adc_interrupt_flag_get`

Function name	<code>adc_interrupt_flag_get</code>
Function prototype	<code>FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, uint32_t int_flag);</code>
Function descriptions	get the ADC interrupt bits
Precondition	-
The called functions	-
Input parameter{in}	
<i>adc_periph</i>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
<i>int_flag</i>	the adc interrupt bits
<i>ADC_INT_FLAG_WDE0</i>	analog watchdog 0 interrupt
<i>ADC_INT_FLAG_EOC</i>	end of group conversion interrupt
<i>ADC_INT_FLAG_EOIC</i>	end of inserted group conversion interrupt
<i>ADC_INT_FLAG_WDE1</i>	analog watchdog 1 interrupt
<i>ADC_INT_FLAG_WDE2</i>	analog watchdog 2 interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```

/* get the ADC0 analog watchdog 0 interrupt bits*/

FlagStatus flag_value;

flag_value = adc_interrupt_flag_get(ADC0, ADC_INT_WDE0);
  
```

adc_interrupt_flag_clear

The description of adc_interrupt_flag_clear is shown as below:

Table 3-47. Function adc_interrupt_flag_clear

Function name	adc_interrupt_flag_clear
Function prototype	void adc_interrupt_flag_clear(uint32_t adc_periph, uint32_t int_flag);
Function descriptions	clear the ADC interrupt bits
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
int_flag	the adc interrupt bits
ADC_INT_FLAG_WDE0	analog watchdog 0 interrupt
ADC_INT_FLAG_EOC	end of group conversion interrupt
ADC_INT_FLAG_EOIC	end of inserted group conversion interrupt
ADC_INT_FLAG_WDE1	analog watchdog 1 interrupt
ADC_INT_FLAG_WDE2	analog watchdog 2 interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* clear the ADC0 analog watchdog 0 interrupt bits*/

adc_interrupt_flag_clear(ADC0, ADC_INT_WDE0);
  
```

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 3-48. BKP Registers

Registers	Descriptions
BKP_DATAx (x=0..41)	Backup data register
BKP_OCTL	RTC signal output control register
BKP_TPCTL	Tamper pin control register
BKP_TPCS	Tamper control and status register

3.3.2. Descriptions of Peripheral functions

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

Table 3-49. BKP firmware function

Function name	Function description
bkp_deinit	reset data registers
bkp_write_data	write BKP data register
bkp_read_data	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, the RTC output can be select as alarm pulse or second pulse
bkp_RTC_clock_output_select	select RTC clock output
bkp_RTC_clock_calibration_direction	select RTC clock calibration direction
bkp_RTC_calibration_value_set	set RTC clock calibration value
bkp_tamper_detection_enable	enable tamper detection
bkp_tamper_detection_disable	disable tamper detection
bkp_tamper_active_level_set	set tamper pin active level
bkp_tamper_interrupt_enable	enable tamper interrupt
bkp_tamper_interrupt_disable	disable tamper interrupt
bkp_flag_get	get bkp flag state
bkp_flag_clear	clear bkp flag state
bkp_interrupt_flag_get	get bkp interrupt flag state
bkp_interrupt_flag_clear	clear bkp interrupt flag state

Enum bkp_data_register_enum
Table 3-50. Enum bkp_data_register_enum

Member name	Function description
BKP_DATA_0	bkp data register number 0
BKP_DATA_1	bkp data register number 1
BKP_DATA_2	bkp data register number 2
BKP_DATA_3	bkp data register number 3
BKP_DATA_4	bkp data register number 4
BKP_DATA_5	bkp data register number 5
BKP_DATA_6	bkp data register number 6
BKP_DATA_7	bkp data register number 7
BKP_DATA_8	bkp data register number 8
BKP_DATA_9	bkp data register number 9
BKP_DATA_10	bkp data register number 10
BKP_DATA_11	bkp data register number 11
BKP_DATA_12	bkp data register number 12
BKP_DATA_13	bkp data register number 13
BKP_DATA_14	bkp data register number 14
BKP_DATA_15	bkp data register number 15
BKP_DATA_16	bkp data register number 16
BKP_DATA_17	bkp data register number 17
BKP_DATA_18	bkp data register number 18
BKP_DATA_19	bkp data register number 19
BKP_DATA_20	bkp data register number 20
BKP_DATA_21	bkp data register number 21
BKP_DATA_22	bkp data register number 22
BKP_DATA_23	bkp data register number 23
BKP_DATA_24	bkp data register number 24
BKP_DATA_25	bkp data register number 25
BKP_DATA_26	bkp data register number 26
BKP_DATA_27	bkp data register number 27
BKP_DATA_28	bkp data register number 28
BKP_DATA_29	bkp data register number 29
BKP_DATA_30	bkp data register number 30
BKP_DATA_31	bkp data register number 31
BKP_DATA_32	bkp data register number 32
BKP_DATA_33	bkp data register number 33
BKP_DATA_34	bkp data register number 34
BKP_DATA_35	bkp data register number 35
BKP_DATA_36	bkp data register number 36
BKP_DATA_37	bkp data register number 37
BKP_DATA_38	bkp data register number 38

Member name	Function description
BKP_DATA_39	bkp data register number 39
BKP_DATA_40	bkp data register number 40
BKP_DATA_41	bkp data register number 41

bkp_deinit

The description of bkp_deinit is shown as below:

Table 3-51. Function bkp_deinit

Function name	bkp_deinit
Function prototype	void bkp_deinit(void);
Function descriptions	reset data 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_write_data

The description of bkp_write_data is shown as below:

Table 3-52. Function bkp_write_data

Function name	bkp_write_data
Function prototype	void bkp_write_data(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 3-50. Enum bkp_data_register_enum
BKP_DATA_x(x = 0..41)	bkp data register number x
Input parameter{in}	
data	the data to be write in BKP data register
0-0xffff	data value

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write BKP data register */

bkp_write_data (BKP_DATA_0, 0x1226);
```

bkp_read_data

The description of bkp_read_data is shown as below:

Table 3-53. Function bkp_data_read

Function name	bkp_read_data
Function prototype	uint16_t bkp_read_data(bkp_data_register_enum register_number);
Function descriptions	read BKP data register
Precondition	-
The called functions	-
Input parameter{in}	
register_number	refer to bkp_data_register_enum
BKP_DATA_x(x = 0..41)	bkp data register number x
Output parameter{out}	
-	-
Return value	
uint16_t	0-0xffff

Example:

```
/* read BKP data register */

uint16_t data;

data = bkp_read_data (BKP_DATA_0);
```

bkp_rtc_calibration_output_enable

The description of bkp_rtc_calibration_output_enable is shown as below:

Table 3-54. 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 3-55. 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 3-56. 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 3-57. 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 3-58. 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, the RTC output can be select as alarm pulse or second pulse
Precondition	-
The called functions	-
Input parameter{in}	
outsels	RTC output selection
RTC_OUTPUT_ALAR	RTC alarm pulse is selected as the RTC output

<i>M_PULSE</i>	
<i>RTC_OUTPUT_SECOND_PULSE</i>	RTC second pulse is selected as the RTC output
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select RTC output alarm signal 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 3-59. 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	select RTC clock output, the RTC clock output can be select as divided 64 or no division
Precondition	-
The called functions	-
Input parameter{in}	
<i>clocksel</i>	RTC clock output selection
<i>RTC_CLOCK_DIV_64</i>	RTC clock divided 64 is selected as the RTC clock output
<i>RTC_CLOCK_DIV_1</i>	RTC clock is selected as the RTC clock output
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select RTC clock devided 64 to output */
bkp_rtc_clock_output_select (RTC_CLOCK_DIV_64);
```

bkp_rtc_clock_calibration_direction

The description of bkp_rtc_clock_calibration_direction is shown as below:

Table 3-60. 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	select RTC clock calibration direction, the RTC clock calibration direction

	can be select as slowed down or speed up
Precondition	-
The called functions	-
Input parameter{in}	
direction	RTC clock calibration direction
<i>RTC_CLOCK_SLOWED_D_DOWN</i>	RTC clock slowed down
<i>RTC_CLOCK_SPEED_UP</i>	RTC clock speed up
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set RTC clock slowed down */

bkp_rtc_clock_calibration_direction (RTC_CLOCK_SLOWED_DOWN);
```

bkp_rtc_calibration_value_set

The description of bkp_rtc_calibration_value_set is shown as below:

Table 3-61. 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
<i>0x00 - 0x7F</i>	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 3-62. Function bkp_tamper_detection_enable

Function name	bkp_tamper_detection_enable
Function prototype	void bkp_tamper_detection_enable (void);
Function descriptions	enable tamper detection
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable tamper pin detection */
bkp_tamper_detection_enable();
```

bkp_tamper_detection_disable

The description of bkp_tamper_detection_disable is shown as below:

Table 3-63. Function bkp_tamper_detection_disable

Function name	bkp_tamper_detection_disable
Function prototype	void bkp_tamper_detection_disable (void);
Function descriptions	disable tamper detection
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable tamper pin detection */
bkp_tamper_detection_disable();
```

bkp_tamper_active_level_set

The description of bkp_tamper_active_level_set is shown as below:

Table 3-64. Function bkp_tamper_active_level_set

Function name	bkp_tamper_active_level_set
----------------------	-----------------------------

Function prototype	void bkp_tamper_active_level_set (uint16_t level);
Function descriptions	set tamper pin active level
Precondition	-
The called functions	-
Input parameter{in}	
level	tamper pin active level
<i>TAMPER_PIN_ACTIVE_HIGH</i>	the tamper pin is active high
<i>TAMPER_PIN_ACTIVE_LOW</i>	the tamper pin is active low
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set tamper pin active level high */
bkp_tamper_active_level_set (TAMPER_PIN_ACTIVE_HIGH);
```

bkp_tamper_interrupt_enable

The description of bkp_tamper_interrupt_enable is shown as below:

Table 3-65. Function bkp_tamper_interrupt_enable

Function name	bkp_tamper_interrupt_enable
Function prototype	void bkp_tamper_interrupt_enable (void);
Function descriptions	enable tamper interrupt
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable tamper pin interrupt */
bkp_tamper_interrupt_enable ();
```

bkp_tamper_interrupt_disable

The description of bkp_tamper_interrupt_disable is shown as below:

Table 3-66. Function bkp_tamper_interrupt_disable

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

Example:

```
/* disable tamper pin interrupt */
bkp_tamper_interrupt_disable();
```

bkp_flag_get

The description of bkp_flag_get is shown as below:

Table 3-67. Function bkp_flag_get

Function name	bkp_flag_get
Function prototype	FlagStatus bkp_flag_get(uint16_t flag);
Function descriptions	get bkp flag state
Precondition	-
The called functions	-
Input parameter{in}	
flag	bkp flag state
BKP_FLAG_TAMPER	tamper event flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get BKP flag state */
FlagStatus status;
status = bkp_flag_get (BKP_FLAG_TAMPER);
```

bkp_flag_clear

The description of bkp_flag_clear is shown as below:

Table 3-68. Function bkp_flag_clear

Function name	bkp_flag_clear
Function prototype	void bkp_flag_clear(uint16_t flag);
Function descriptions	clear bkp flag state
Precondition	-
The called functions	-
Input parameter{in}	
flag	bkp flag state
BKP_FLAG_TAMPER	tamper event flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear BKP flag state */

bkp_flag_clear (BKP_FLAG_TAMPER);
```

bkp_interrupt_flag_get

The description of bkp_interrupt_flag_get is shown as below:

Table 3-69. 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 state
Precondition	-
The called functions	-
Input parameter{in}	
flag	bkp interrupt flag state
BKP_INT_FLAG_TAMP ER	tamper interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get BKP interrupt flag state */

bkp_interrupt_flag_get (BKP_INT_FLAG_TAMPER);
```

bkp_interrupt_flag_clear

The description of bkp_interrupt_flag_clear is shown as below:

Table 3-70. 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 state
Precondition	-
The called functions	-
Input parameter{in}	
flag	bkp interrupt flag state
<i>BKP_INT_FLAG_TAMP ER</i>	tamper interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear BKP interrupt flag state */
bkp_interrupt_flag_clear (BKP_INT_FLAG_TAMPER);
```

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 3-71. 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

Registers	Descriptions
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
CAN_RFIFOMDAT A1x	Receive FIFO mailbox data1 register
CAN_FCTL	Filter control register
CAN_FMCFG	Filter mode configuration register
CAN_FSCFG	Filter scale configuration register
CAN_FA FIFO	Filter associated FIFO register
CAN_FW	Filter working register
CAN_FxDATAy	Filter x data y register

Table 3-72. CAN-FD 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_FDCTL	FD control register
CAN_FDSTAT	FD status register
CAN_FDTDC	FD transmitter delay compensation register
CAN_DBT	Date 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
CAN_RFIFOMDAT A1x	Receive FIFO mailbox data1 register
CAN_FCTL	Filter control register

Registers	Descriptions
CAN_FMCFG	Filter mode configuration register
CAN_FSCFG	Filter scale configuration register
CAN_FAFIFO	Filter associated FIFO register
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 3-73. CAN firmware function

Function name	Function description
can_deinit	deinitialize CAN
can_init	initialize CAN
can_filter_init	initialize CAN filter
can_filter_mask_mode_init	CAN filter mask mode initialization
can_struct_para_init	initialize CAN parameter struct with a default value
can_monitor_mode_set	CAN communication mode configure
can_fd_init	initialize CAN FD function
can_fd_function_enable	CAN FD frame function enable
can_fd_function_disable	CAN FD frame function disable
can1_filter_start_bank	set can1 filter start bank number
can_debug_freeze_enable	CAN debug freeze enable
can_debug_freeze_disable	CAN debug freeze disable
can_time_trigger_mode_enable	CAN time trigger mode enable
can_time_trigger_mode_disable	CAN time trigger mode disable
can_message_transmit	transmit CAN message
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

Function name	Function description
can_interrupt_flag_clear	CAN clear interrupt flag state

Structure can_parameter_struct

Table 3-74. can_parameter_struct

Member name	Function description
working_mode	CAN working mode
resync_jump_width	CAN resynchronization jump width
time_segment_1	time segment 1
time_segment_2	time segment 2
time_triggered	time triggered communication mode
auto_bus_off_recovery	automatic bus-off recovery
auto_wake_up	automatic wake-up mode
auto_retrans	automatic retransmission mode
rec_fifo_overwrite	receive FIFO overwrite mode
trans_fifo_order	transmit FIFO order
prescaler	baudrate prescaler

Structure can_trasnmit_message_struct

Table 3-75. can_trasnmit_message_struct

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

Table 3-76. can_trasnmit_message_struct (for CAN-FD)

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[64]	transmit data
fd_flag	CAN FD frame flag
fd_brs	bit rate of data switch
fd_esi	error status indicator

Structure can_receive_message_struct

Table 3-77. can_receive_message_struct

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

Table 3-78. can_receive_message_struct (for CAN-FD)

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[64]	receive data
rx_hi	filtering index
fd_flag	CAN FD frame flag
fd_brs	bit rate of data switch
fd_esi	error status indicator

Structure can_fd_tdc_struct

Table 3-79. can_fd_tdc_struct (for CAN-FD)

Member name	Function description
tdc_mode	transmitter delay compensation mode
tdc_filter	transmitter delay compensation filter
tdc_offset	transmitter delay compensation offset

Structure can_fdframe_struct

Table 3-80. can_fdframe_struct (for CAN-FD)

Member name	Function description
fd_frame	FD operation function
excp_event_detect	protocol exception event detection function
delay_compensation	transmitter delay compensation mode
p_delay_compensation	pointer to the struct of the transmitter delay compensation, refer to Table 3-79. can_fd_tdc_struct (for CAN-FD) .
iso_bosch	ISO/Bosch mode choice
esi_mode	error state indicator mode

data_resync_jump_width	CAN resynchronization jump width
data_time_segment_1	time segment 1
data_time_segment_2	time segment 2
data_prescaler	baudrate prescaler

Structure can_filter_parameter_struct

Table 3-81. 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_interrupt_flag_enum

Table 3-82. Enum can_interrupt_flag_enum

enum name	enum description
CAN_INT_FLAG_SLPIF	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_ERRIF	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_RFL0	receive FIFO0 not empty interrupt flag
CAN_INT_FLAG_RFO1	receive FIFO1 overfull interrupt flag
CAN_INT_FLAG_RFF1	receive FIFO1 full interrupt flag
CAN_INT_FLAG_RFL1	receive FIFO1 not empty interrupt flag
CAN_INT_FLAG_ERRN	error number interrupt flag
CAN_INT_FLAG_BOERR	bus-off error interrupt flag
CAN_INT_FLAG_PERR	passive error interrupt flag
CAN_INT_FLAG_WERR	warning error interrupt flag

Enum can_flag_enum
Table 3-83. Enum can_flag_enum

enum name	enum description
CAN_FLAG_RXL	CAN flags
CAN_FLAG_LASTRX	RX level
CAN_FLAG_RS	last sample value of RX pin
CAN_FLAG_TS	receiving state
CAN_FLAG_SLPIF	transmitting state
CAN_FLAG_WUIF	status change flag of entering sleep working mode
CAN_FLAG_ERRIF	status change flag of wakeup from sleep working mode
CAN_FLAG_SLPWS	error flag
CAN_FLAG_IWS	sleep 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_MTFNERR 2	mailbox 2 transmit finished with no error
CAN_FLAG_MTFNERR 1	mailbox 1 transmit finished with no error
CAN_FLAG_MTFNERR 0	mailbox 0 transmit finished with no error
CAN_FLAG_MTF2	mailbox 2 transmit finished
CAN_FLAG_MTF1	mailbox 1 transmit finished
CAN_FLAG_MTF0	mailbox 0 transmit finished
CAN_FLAG_RFO0	receive FIFO0 overfull
CAN_FLAG_RFF0	receive FIFO0 full
CAN_FLAG_RFO1	receive FIFO1 overfull
CAN_FLAG_RFF1	receive FIFO1 full
CAN_FLAG_BOERR	bus-off error
CAN_FLAG_PERR	passive error
CAN_FLAG_WERR	warning error

Enum can_error_enum

Table 3-84. Enum can_error_enum

enum name	enum description
CAN_ERROR_NONE	no error
CAN_ERROR_FILL	fill error
CAN_ERROR_FORMATE	format error
CAN_ERROR_ACK	ACK error
CAN_ERROR_BITRECESSIVE	bit recessive error
CAN_ERROR_BITDOMINANTER	bit dominant error
CAN_ERROR_CRC	CRC error
CAN_ERROR_SOFTWARECFG	software configure

Enum can_transmit_state_enum

Table 3-85. Enum can_transmit_state_enum

enum name	enum 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_format_fifo_enum

Table 3-86. Enum can_format_fifo_enum

enum name	enum description
CAN_STANDARD_FIFO0	standard frame and used FIFO0
CAN_STANDARD_FIFO1	standard frame and used FIFO1
CAN_EXTENDED_FIFO0	extended frame and used FIFO0
CAN_EXTENDED_FIFO1	extended frame and used FIFO1

Enum can_struct_type_enum

Table 3-87. Enum can_struct_type_enum

enum name	enum description
CAN_INIT_STRUCT	CAN initiliaz parameters struct
CAN_FILTER_STRUCT	CAN filter 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 3-88. 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=0,1,2)	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 3-89. Function can_struct_para_init

Function name	can_struct_para_init
Function prototype	void can_struct_para_init(can_struct_type_enum type, void* p_struct)
Function descriptions	initialize CAN parameter struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
type	CAN peripheral refer to Table 3-87. Enum can_struct_type_enum
CAN_INIT_STRUCT	CAN initilize parameters struct
CAN_FILTER_STRUCT	CAN filter parameters struct
CAN_FD_FRAME_STRUCT	CAN initilize FD frame parameters struct
CAN_TX_MESSAGE_STRUCT	CAN transmit message struct
CAN_RX_MESSAGE_STRUCT	CAN receive message struct
Output parameter{out}	
p_struct	the struct pointer that needs initialize
Return value	
-	-

Example:

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

can_init

The description of can_init is shown as below:

Table 3-90. Function can_init

Function name	can_init
Function prototype	ErrStatus can_init(uint32_t can_periph, can_parameter_struct* can_parameter_init);
Function descriptions	initialize CAN
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
Input parameter{in}	
can_parameter_init	CAN parameter initialization stuct, the structure members can refer to members of the structure Table 3-74. can_parameter_struct
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR

Example:

```
/* CAN0 initialize*/
can_parameter_struct can_parameter_init;
can_init (CAN0, &can_parameter_init);
```

can_filter_init

The description of can_filter_init is shown as below:

Table 3-91. Function can_filter_init

Function name	can_filter_init
Function prototype	void can_filter_init(can_filter_parameter_struct* can_filter_parameter_init);
Function descriptions	initialize CAN filter
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral

CANx(x=0,2)	CAN peripheral selection
Input parameter{in}	
can_filter_parameter_i	CAN filter initialization struct, the structure members can refer to members of the structure Table 3-81. can_filter_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize CAN filter */

can_filter_init(CAN0, &can_filter);
```

can_filter_mask_mode_init

The description of can_filter_mask_mode_init is shown as below:

Table 3-92. Function can_filter_mask_mode_init

Function name	can_filter_mask_mode_init
Function prototype	void can_filter_mask_mode_init(uint32_t id, uint32_t mask, can_format_fifo_enum format_fifo, uint16_t filter_number)
Function descriptions	CAN filter mask mode initialization
Precondition	-
The called functions	can_filter_init()
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,2)	CAN peripheral selection
Input parameter{in}	
id	value range (0x00000000 - 0x1FFFFFFF)
Input parameter{in}	
mask	value range (0x00000000 - 0x1FFFFFFF)
Input parameter{in}	
format_fifo	format and fifo states refer to Table 3-86. Enum can_format_fifo_enum , only one parameter can be selected which is shown as below
CAN_STANDARD_FIF 00	standard format and store to FIFO0
CAN_STANDARD_FIF 01	standard format and store to FIFO1
CAN_EXTENDED_FIF 00	extended format and store to FIFO0
CAN_EXTENDED_FIF 01	extended format and store to FIFO1
Input parameter{in}	

filter_number	filter sequence number, value range(0x00 - 0x1C)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
can_filter_mask_mode_init(CAN0, 0x11, 0x11, CAN_STANDARD_FIFO0, 0);
```

can_monitor_mode_set

The description of can_monitor_mode_set is shown as below:

Table 3-93. Function can_monitor_mode_set

Function name	can_monitor_mode_set
Function prototype	ErrStatus can_monitor_mode_set(uint32_t can_periph, uint8_t mode)
Function descriptions	CAN communication mode configure
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
Input parameter{in}	
mode	communication mode, only one parameter can be selected which is shown as below
CAN_NORMAL_MODE	normal mode
CAN_LOOPBACK_MODE	loopback mode
CAN_SILENT_MODE	silent mode
CAN_SILENT_LOOPBACK_MODE	silent loopback mode
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR

Example:

```
can_monitor_mode_set(CAN0, CAN_NORMAL_MODE);
```

can_fd_init (for CAN-FD)

The description of can_fd_init is shown as below:

Table 3-94. Function can_fd_init

Function name	can_fd_init
---------------	-------------

Function prototype	ErrStatus can_fd_init(uint32_t can_periph, can_fdframe_struct* can_fdframe_init);
Function descriptions	initialize CAN FD function
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
Input parameter{in}	
can_fdframe_init	parameters for CAN FD initialization, the structure members can refer to members of the structure Table 3-79. can_fd_tdc_struct (for CAN-FD)
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR

Example:

```
/* CAN0 FD initialize*/
can_fdframe_struct fd_init_para;
can_fd_init(CAN0, &fd_init_para);
```

can_fd_function_enable (for CAN-FD)

The description of can_fd_function_enable is shown as below:

Table 3-95. Function can_fd_function_enable

Function name	can_fd_function_enable
Function prototype	void can_fd_function_enable(uint32_t can_periph)
Function descriptions	CAN FD frame function enable
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
can_fd_function_enable(CAN0);
```

can_fd_function_disable (for CAN-FD)

The description of can_fd_function_disable is shown as below:

Table 3-96. Function can_fd_function_disable

Function name	can_fd_function_disable
Function prototype	void can_fd_function_disable(uint32_t can_periph)
Function descriptions	CAN FD frame function disable
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
can_fd_function_disable(CAN0);
```

can1_filter_start_bank

The description of can1_filter_start_bank is shown as below:

Table 3-97. 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 filter start bank number
Precondition	-
The called functions	-
Input parameter{in}	
start_bank	CAN1 start bank number
1..27	start number
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set CAN1 filter start bank number 15*/
```

```
can1_filter_start_bank (15);
```

can_debug_freeze_enable

The description of can_debug_freeze_enable is shown as below:

Table 3-98. Function can_debug_freeze_enable

Function name	can_debug_freeze_enable
Function prototype	void can_debug_freeze_enable(uint32_t can_periph);
Function descriptions	enable CAN debug freeze
Precondition	-
The called functions	dbg_periph_enable
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	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 3-99. Function can_debug_freeze_disable

Function name	can_debug_freeze_disable
Function prototype	void can_debug_freeze_disable(uint32_t can_periph);
Function descriptions	disable CAN debug freeze
Precondition	-
The called functions	dbg_periph_disable
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	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 3-100. Function can_time_trigger_mode_enable

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

Example:

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

can_time_trigger_mode_disable

The description of can_time_trigger_mode_disable is shown as below:

Table 3-101. Function can_time_trigger_mode_disable

Function name	can_time_trigger_mode_disable
Function prototype	void can_time_trigger_mode_disable(uint32_t can_periph);
Function descriptions	disable CAN time trigger mode
Precondition	-
The called functions	-

Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	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 3-102. Function can_message_transmit

Function name	can_message_transmit
Function prototype	uint8_t can_message_transmit(uint32_t can_periph, can_transmit_message_struct* transmit_message);
Function descriptions	transmit CAN message
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
Input parameter{in}	
transmit_message	CAN transmit message stuct, the structure members refer to Table 3-75. can_transmit_message_struct CAN-FD transmit message stuct, the structure members refer to Table 3-76. can_transmit_message_struct (for CAN-FD)
Output parameter{out}	
-	-
Return value	
uint8_t	0x00-0x03

Example:

```
/* CAN0 transmit message and return the mailbox number*/

uint8_t transmit_mailbox = 0;

transmit_mailbox = can_message_transmit(CAN0, &transmit_message);
```

can_transmit_states

The description of can_transmit_states is shown as below:

Table 3-103. 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=0,1,2)	CAN peripheral selection
Input parameter{in}	
mailbox_number	Mailbox number
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
Output parameter{out}	
-	-
Return value	
can_transmit_state_enum num	the return value refer to Table 3-85. Enum can_transmit_state_enum

Example:

```
/* CAN0 mailbox0 transmit state */
uint8_t transmit_state = 0;
transmit_state = can_transmit_states (CAN0, CAN_MAILBOX0);
```

can_transmission_stop

The description of can_transmission_stop is shown as below:

Table 3-104. Function can_transmission_stop

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

CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* stop CAN0 mailbox0 transmission */

can_transmission_stop (CAN0, CAN_MAILBOX0);
```

can_message_receive

The description of can_message_receive is shown as below:

Table 3-105. Function can_message_receive

Function name	can_message_receive
Function prototype	void can_message_receive(uint32_t can_periph, uint8_t fifo_number, can_receive_message_struct* receive_message);
Function descriptions	CAN receive message
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
Input parameter{in}	
fifo_number	Fifo number
CAN_FIFOx	CAN_FIFOx(x=0,1)
Input parameter{in}	
receive_message	CAN message receive stuct, the structure members refer to Table 3-77. can_receive_message_struct CAN-FD message receive stuct, the structure members refer to Table 3-78. can_receive_message_struct (for CAN-FD)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CAN0 FIFO0 receive message */

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

can_fifo_release

The description of can_fifo_release is shown as below:

Table 3-106. Function can_fifo_release

Function name	can_fifo_release
Function prototype	void can_fifo_release(uint32_t can_periph, uint8_t fifo_number);
Function descriptions	release FIFO0
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
Input parameter{in}	
fifo_number	Fifo number
CAN_FIFOx	CAN_FIFOx(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CAN0 release FIFO0*/
can_fifo_release (CAN0, CAN_FIFO0);
```

can_receive_message_length_get

The description of can_receive_message_length_get is shown as below:

Table 3-107. Function can_receive_message_length_get

Function name	can_receive_message_length_get
Function prototype	uint8_t can_receive_message_length_get(uint32_t can_periph, uint8_t fifo_number);
Function descriptions	CAN receive message length
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
Input parameter{in}	
fifo_number	Fifo number
CAN_FIFOx	CAN_FIFOx(x=0,1)
Output parameter{out}	
-	-

Return value	
uint8_t	0..3

Example:

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

uint8_t frame_number = 0;

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

can_working_mode_set

The description of can_working_mode_set is shown as below:

Table 3-108. 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=0,1,2)	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 3-109. Function can_wakeup

Function name	can_wakeup
---------------	------------

Function prototype	ErrStatus can_wakeup(uint32_t can_periph);
Function descriptions	wake up CAN
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	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 3-110. Function can_error_get

Function name	can_error_get
Function prototype	can_error_enum can_error_get(uint32_t can_periph);
Function descriptions	get CAN error type
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
can_error_enum	0..7

Example:

```
/* get CAN0 error type */
can_error_get (CAN0);
```

can_receive_error_number_get

The description of can_receive_error_number_get is shown as below:

Table 3-111. Function can_receive_error_number_get

Function name	can_receive_error_number_get
----------------------	------------------------------

Function prototype	uint8_t can_receive_error_number_get(uint32_t can_periph);
Function descriptions	get CAN receive error number
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
uint8_t	0..255

Example:

```
/* get CAN0 receive error number */
can_receive_error_number_get (CAN0);
```

can_transmit_error_number_get

The description of `can_transmit_error_number_get` is shown as below:

Table 3-112. Function can_transmit_error_number_get

Function name	can_transmit_error_number_get
Function prototype	uint8_t can_transmit_error_number_get(uint32_t can_periph);
Function descriptions	get CAN transmit error number
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
uint8_t	the return value refer to Table 3-84. Enum can_error_enum

Example:

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

can_interrupt_enable

The description of `can_interrupt_enable` is shown as below:

Table 3-113. Function can_interrupt_enable

Function name	can_interrupt_enable
----------------------	----------------------

Function prototype	void can_interrupt_enable(uint32_t can_periph, uint32_t interrupt);
Function descriptions	enable CAN interrupt
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
Input parameter{in}	
interrupt	Interrupt type
CAN_INT_TME	transmit mailbox empty interrupt enable
CAN_INT_RFNE0	receive FIFO0 not empty interrupt enable
CAN_INT_RFF0	receive FIFO0 full interrupt enable
CAN_INT_RF00	receive FIFO0 overfull interrupt enable
CAN_INT_RFNE1	receive FIFO1 not empty interrupt enable
CAN_INT_RFF1	receive FIFO1 full interrupt enable
CAN_INT_RF01	receive FIFO1 overfull interrupt enable
CAN_INT_WERR	warning error interrupt enable
CAN_INT_PERR	passive error interrupt enable
CAN_INT_BO	bus-off interrupt enable
CAN_INT_ERRN	error number interrupt enable
CAN_INT_ERR	error interrupt enable
CAN_INT_WU	wakeup interrupt enable
CAN_INT_SLPW	sleep working interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

can_interrupt_disable

The description of can_interrupt_disable is shown as below:

Table 3-114. 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
CANx(x=0,1,2)	CAN peripheral selection
Input parameter{in}	
interrupt	Interrupt type
CAN_INT_TME	transmit mailbox empty interrupt enable
CAN_INT_RFNE0	receive FIFO0 not empty interrupt enable
CAN_INT_RFF0	receive FIFO0 full interrupt enable
CAN_INT_RFO0	receive FIFO0 overfull interrupt enable
CAN_INT_RFNE1	receive FIFO1 not empty interrupt enable
CAN_INT_RFF1	receive FIFO1 full interrupt enable
CAN_INT_RFO1	receive FIFO1 overfull interrupt enable
CAN_INT_WERR	warning error interrupt enable
CAN_INT_PERR	passive error interrupt enable
CAN_INT_BO	bus-off interrupt enable
CAN_INT_ERRN	error number interrupt enable
CAN_INT_ERR	error interrupt enable
CAN_INT_WU	wakeup interrupt enable
CAN_INT_SLPW	sleep working interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

can_flag_get

The description of can_flag_get is shown as below:

Table 3-115. Function can_flag_get

Function name	can_flag_get
Function prototype	FlagStatus can_flag_get(uint32_t can_periph, can_flag_enum flag);
Function descriptions	get CAN flag state
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection
Input parameter{in}	
flag	CAN flags refer to Table 3-83. Enum can_flag_enum
CAN_FLAG_RXL	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 R2</code>	mailbox 2 transmit finished with no error
<code>CAN_FLAG_MTFNER R1</code>	mailbox 1 transmit finished with no error
<code>CAN_FLAG_MTFNER 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_RF00</code>	receive FIFO0 overfull
<code>CAN_FLAG_RFF0</code>	receive FIFO0 full
<code>CAN_FLAG_RF01</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	
<code>FlagStatus</code>	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 3-116. Function can_flag_clear

Function name	can_flag_clear
Function prototype	void can_flag_clear(uint32_t can_periph, can_flag_enum flag);
Function descriptions	clear CAN flag state
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1,2)	CAN peripheral selection, the CAN1 only for GD32F30X_CL
Input parameter{in}	
flag	CAN flags refer to Table 3-83. Enum can_flag_enum
CAN_FLAG_MTE2	mailbox 2 transmit error
CAN_FLAG_MTE1	mailbox 1 transmit error
CAN_FLAG_MTE0	mailbox 0 transmit error
CAN_FLAG_MTF2	mailbox 2 transmit finished
CAN_FLAG_MTF1	mailbox 1 transmit finished
CAN_FLAG_MTF0	mailbox 0 transmit finished
CAN_FLAG_RFO0	receive FIFO0 overfull
CAN_FLAG_RFF0	receive FIFO0 full
CAN_FLAG_RFO1	receive FIFO1 overfull
CAN_FLAG_RFF1	receive FIFO1 full
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

can_interrupt_flag_get

The description of can_interrupt_flag_get is shown as below:

Table 3-117. 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=0,1,2)		CAN peripheral selection
Input parameter{in}		
flag		CAN interrupt flags refer to Table 3-82. Enum can_interrupt_flag enum
<i>CAN_INT_FLAG_SLPI_F</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_RF00</i>		receive FIFO0 overfull interrupt flag
<i>CAN_INT_FLAG_RFF0</i>		receive FIFO0 full interrupt flag
<i>CAN_INT_FLAG_RF01</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
Output parameter{out}		
-		-
Return value		
FlagStatus	SET / RESET	

Example:

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

can_interrupt_flag_clear

The description of can_interrupt_flag_clear is shown as below:

Table 3-118. 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
CANx(x=0,1,2)	CAN peripheral selection
Input parameter{in}	
flag	CAN interrupt flags refer to Table 3-82. Enum can_interrupt_flag_enum
CAN_INT_FLAG_SLP1_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_ERR1_F	error interrupt flag
CAN_INT_FLAG_MTF2	mailbox 2 transmit finished interrupt flag
CAN_INT_FLAG_MTF1	mailbox 1 transmit finished interrupt flag
CAN_INT_FLAG_MTF0	mailbox 0 transmit finished interrupt flag
CAN_INT_FLAG_RF00	receive FIFO0 overfull interrupt flag
CAN_INT_FLAG_RFF0	receive FIFO0 full interrupt flag
CAN_INT_FLAG_RF01	receive FIFO1 overfull interrupt flag
CAN_INT_FLAG_RFF1	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. CRC

A cycle 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 [3.5.1](#), the CRC firmware functions are introduced in chapter [3.5.2](#).

3.5.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

Table 3-119. CRC Registers

Registers	Descriptions
CRC_DATA	CRC data register
CRC_FDATA	CRC free data register
CRC_CTL	CRC control register
CRC_IDATA	CRC initialization data register
CRC_POLY	CRC polynomial register

3.5.2. Descriptions of Peripheral functions

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

Table 3-120. CRC firmware function

Function name	Function description
crc_deinit	deinitialize CRC calculation unit
crc_data_register_reset	reset data register to the value of initialization data register
crc_reverse_output_data_enable	enable the reverse operation of output data
crc_reverse_output_data_disable	disable the reverse operation of output data
crc_input_data_reverse_config	configure the CRC input data function
crc_data_register_read	read the data register
crc_free_data_register_read	read the free data register
crc_free_data_register_write	write the free data register
crc_init_data_register_write	write the initialization value register
crc_polynomial_size_set	configure the CRC size of polynomial function
crc_polynomial_set	configure the CRC polynomial value function
crc_single_data_calculate	CRC calculate a 32-bit data
crc_block_data_calculate	CRC calculate a 32-bit data array

crc_deinit

The description of crc_deinit is shown as below:

Table 3-121. Function crc_deinit

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

Example:

```
/* deinitialize CRC */
```

```
crc_deinit();
```

crc_data_register_reset

The description of `crc_data_register_reset` is shown as below:

Table 3-122. 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 initialaiton data register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset crc data register */
```

```
crc_data_register_reset();
```

crc_reverse_output_data_enable

The description of `crc_reverse_output_data_enable` is shown as below:

Table 3-123. Function `crc_reverse_output_data_enable`

Function name	crc_reverse_output_data_enable
Function prototype	void crc_reverse_output_data_enable (void);
Function descriptions	enable the reverse operation of output data
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable CRC reverse operation of output data */
```

```
crc_reverse_output_data_enable();
```

crc_reverse_output_data_disable

The description of `crc_reverse_output_data_disable` is shown as below:

Table 3-124. Function `crc_reverse_output_data_disable`

Function name	crc_reverse_output_data_disable
Function prototype	void crc_reverse_output_data_disable (void);
Function descriptions	disable the reverse operation of output data
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable crc reverse operation of output data */
```

```
crc_reverse_output_data_disable();
```

crc_input_data_reverse_config

The description of `crc_input_data_reverse_config` is shown as below:

Table 3-125. Function `crc_input_data_reverse_config`

Function name	crc_input_data_reverse_config
Function prototype	void crc_input_data_reverse_config(uint32_t data_reverse)
Function descriptions	configure the crc input data function
Precondition	-
The called functions	-
Input parameter{in}	
data_reverse	specify input data reverse function
CRC_INPUT_DATA_N OT	input data is not reversed
CRC_INPUT_DATA_B YTE	input data is reversed on 8 bits
CRC_INPUT_DATA_H ALFWORD	input data is reversed on 16 bits
CRC_INPUT_DATA_W ORD	input data is reversed on 32 bits

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the crc input data */
crc_input_data_reverse_config (CRC_INPUT_DATA_WORD);
```

crc_data_register_read

The description of `crc_data_register_read` is shown as below:

Table 3-126. 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 data register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	32-bit value of the data register (0-0xFFFFFFFF)

Example:

```
/* read crc data register */
uint32_t crc_value = 0;
crc_value = crc_data_register_read();
```

crc_free_data_register_read

The description of `crc_free_data_register_read` is shown as below:

Table 3-127. 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 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 3-128. Function `crc_free_data_register_write`

Function name	crc_free_data_register_write
Function prototype	void crc_free_data_register_write(uint8_t free_data);
Function descriptions	write 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_init_data_register_write

The description of `crc_init_data_register_write` is shown as below:

Table 3-129. Function `crc_init_data_register_write`

Function name	crc_init_data_register_write
Function prototype	void crc_init_data_register_write(uint32_t init_data)
Function descriptions	write the initializaiton data register
Precondition	-
The called functions	-
Input parameter{in}	
init_data	specify 32-bit data

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write crc initializaiton data register */

crc_init_data_register_write (0x11223344);
```

crc_polynomial_size_set

The description of `crc_polynomial_size_set` is shown as below:

Table 3-130. Function `crc_polynomial_size_set`

Function name	crc_polynomial_size_set
Function prototype	void crc_polynomial_size_set(uint32_t poly_size)
Function descriptions	configure the CRC size of polynomial function
Precondition	-
The called functions	-
Input parameter{in}	
poly_size	size of polynomial
CRC_CTL_PS_32	32-bit polynomial for CRC calculation
CRC_CTL_PS_16	16-bit polynomial for CRC calculation
CRC_CTL_PS_8	8-bit polynomial for CRC calculation
CRC_CTL_PS_7	7-bit polynomial for CRC calculation
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the CRC polynomial size*/

crc_polynomial_size_set (CRC_CTL_PS_7);
```

crc_polynomial_set

The description of `crc_polynomial_set` is shown as below:

Table 3-131. Function `crc_polynomial_set`

Function name	crc_polynomial_set
Function prototype	void crc_polynomial_set(uint32_t poly)
Function descriptions	configure the CRC polynomial value function
Precondition	-
The called functions	-

Input parameter{in}	
poly	configurable polynomial value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the CRC polynomial value */
crc_polynomial_set (0x11223344);
```

crc_single_data_calculate

The description of `crc_single_data_calculate` is shown as below:

Table 3-132. Function `crc_single_data_calculate`

Function name	crc_single_data_calculate
Function prototype	uint32_t crc_single_data_calculate(uint32_t sdata);
Function descriptions	CRC calculate 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 3-133. 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};

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

3.6. CTC

The CTC unit trims the frequency of the IRC48M which is based on an external accurate reference signal source. It can adjust the calibration value to provide a precise IRC48M clock automatically or manually. The CTC registers are listed in chapter [3.6.1](#), the CTC firmware functions are introduced in chapter [3.6.2](#).

3.6.1. Descriptions of Peripheral registers

CTC registers are listed in the table shown as below:

Table 3-134. CTC Registers

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

3.6.2. Descriptions of Peripheral functions

CTC registers are listed in the table shown as below:

Table 3-135. CTC firmware function

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

ctc_deinit

The description of ctc_deinit is shown as below:

Table 3-136. Function ctc_deinit

Function name	ctc_deinit
Function prototype	void ctc_deinit (void)
Function descriptions	Reset CTC peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset CTC */

ctc_deinit();
```

ctc_counter_enable

The description of `ctc_counter_enable` is shown as below:

Table 3-137. Function `ctc_counter_enable`

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

Example:

```
/* enable CTC trim counter*/

ctc_counter_enable();
```

ctc_counter_disable

The description of `ctc_counter_disable` is shown as below:

Table 3-138. Function `ctc_counter_disable`

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

Return value	
-	-

Example:

```
/* disable CTC trim counter */

ctc_counter_disable();
```

ctc_irc48m_trim_value_config

The description of `ctc_irc48m_trim_value_config` is shown as below:

Table 3-139. Function `ctc_irc48m_trim_value_config`

Function name	ctc_irc48m_trim_value_config
Function prototype	void ctc_irc48m_trim_value_config(uint8_t trim_value);
Function descriptions	configure the IRC48M trim value
Precondition	-
The called functions	-
Input parameter{in}	
trim_value	0~63
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* IRC48M trim value configuration */

ctc_irc48m_trim_value_config (0x01);
```

ctc_software_refsource_pulse_generate

The description of `ctc_software_refsource_pulse_generate` is shown as below:

Table 3-140. Function `ctc_software_refsource_pulse_generate`

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

Example:

```
/* generate reference source sync pulse */
ctc_software_refsource_pulse_generate();
```

ctc_hardware_trim_mode_config

The description of `ctc_hardware_trim_mode_config` is shown as below:

Table 3-141. Function `ctc_hardware_trim_mode_config`

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

Example:

```
/* enable CTC hardware trim */
ctc_hardware_trim_mode_config (CTC_HARDWARE_TRIM_MODE_ENABLE);
```

ctc_refsource_polarity_config

The description of `ctc_refsource_polarity_config` is shown as below:

Table 3-142. Function `ctc_refsource_polarity_config`

Function name	ctc_refsource_polarity_config
Function prototype	void ctc_refsource_polarity_config(uint32_t polarity);
Function descriptions	configure reference signal source polarity
Precondition	-
The called functions	-
Input parameter{in}	
polarity	reference signal source polarity
CTC_REFRESOURCE_POLARITY_FALLING	reference signal source polarity is falling edge

<code>CTC_REFRESOURCE_POLARITY_RISING</code>	reference signal source polarity is rising edge
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

ctc_refresource_signal_select

The description of `ctc_refresource_signal_select` is shown as below:

Table 3-143. Function `ctc_refresource_signal_select`

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

Example:

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

ctc_refresource_prescaler_config

The description of `ctc_refresource_prescaler_config` is shown as below:

Table 3-144. Function `ctc_refresource_prescaler_config`

Function name	<code>ctc_refresource_prescaler_config</code>
Function prototype	<code>void ctc_refresource_prescaler_config(uint32_t prescaler);</code>
Function descriptions	configure reference signal source prescaler

Precondition	-
The called functions	-
Input parameter{in}	
prescaler	Prescaler factor
<i>CTC_REFRESOURCE_P</i> SC_OFF	reference signal not divided
<i>CTC_REFRESOURCE_P</i> SC_DIV2	reference signal divided by 2
<i>CTC_REFRESOURCE_P</i> SC_DIV4	reference signal divided by 4
<i>CTC_REFRESOURCE_P</i> SC_DIV8	reference signal divided by 8
<i>CTC_REFRESOURCE_P</i> SC_DIV16	reference signal divided by 16
<i>CTC_REFRESOURCE_P</i> SC_DIV32	reference signal divided by 32
<i>CTC_REFRESOURCE_P</i> SC_DIV64	reference signal divided by 64
<i>CTC_REFRESOURCE_P</i> SC_DIV128	reference signal divided by 128
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

ctc_clock_limit_value_config

The description of `ctc_clock_limit_value_config` is shown as below:

Table 3-145. Function `ctc_clock_limit_value_config`

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

Return value	
-	-

Example:

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

ctc_counter_reload_value_config

The description of `ctc_counter_reload_value_config` is shown as below:

Table 3-146. Function `ctc_counter_reload_value_config`

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

Example:

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

ctc_counter_capture_value_read

The description of `ctc_counter_capture_value_read` is shown as below:

Table 3-147. Function `ctc_counter_capture_value_read`

Function name	ctc_counter_capture_value_read
Function prototype	uint16_t ctc_counter_capture_value_read(void);
Function descriptions	read CTC counter capture value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	the 16-bit CTC counter capture value (0x0000 - 0xFFFF)

Example:

```
/* read CTC counter capture value */

uint16_t ctc_value = 0;

ctc_value = ctc_counter_capture_value_read();
```

ctc_counter_direction_read

The description of `ctc_counter_direction_read` is shown as below:

Table 3-148. Function `ctc_counter_direction_read`

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

Example:

```
/* read ctc counter direction */

FlagStatus ctc_direction = SET;

ctc_direction = ctc_counter_direction_read();
```

ctc_counter_reload_value_read

The description of `ctc_counter_reload_value_read` is shown as below:

Table 3-149. Function `ctc_counter_reload_value_read`

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

uint16_t	Read 16-bit data of counter reload value (0x0000 - 0xFFFF)
----------	--

Example:

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

uint16_t ctc_reload_value = 0;

ctc_reload_value = ctc_counter_reload_value_read();
```

ctc_ir48m_trim_value_read

The description of ctc_ir48m_trim_value_read is shown as below:

Table 3-150. Function ctc_ir48m_trim_value_read

Function name	ctc_ir48m_trim_value_read
Function prototype	uint8_t ctc_ir48m_trim_value_read(void);
Function descriptions	read the IRC48M trim value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint8_t	the 8-bit IRC48M trim value (0-63)

Example:

```
/* read the IRC48M trim value */

uint8_t ctc_trim_value = 0;

ctc_trim_value = ctc_ir48m_trim_value_read();
```

ctc_flag_get

The description of ctc_flag_get is shown as below:

Table 3-151. Function ctc_flag_get

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

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

Example:

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

ctc_flag_clear

The description of `ctc_flag_clear` is shown as below:

Table 3-152. Function `ctc_flag_clear`

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

Example:

```
/* clear CTC flag status */
ctc_flag_clear (CTC_FLAG_CKOK);
```

ctc_interrupt_enable

The description of ctc_interrupt_enable is shown as below:

Table 3-153. Function ctc_interrupt_enable

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

Example:

```
/* enable CTC clock trim OK interrupt */
ctc_interrupt_enable (CTC_INT_CKOK);
```

ctc_interrupt_disable

The description of ctc_interrupt_disable is shown as below:

Table 3-154. Function ctc_interrupt_disable

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

Example:

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

ctc_interrupt_flag_get

The description of `ctc_interrupt_flag_get` is shown as below:

Table 3-155. Function `ctc_interrupt_flag_get`

Function name	ctc_interrupt_flag_get
Function prototype	FlagStatus ctc_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get CTC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
<code>int_flag</code>	CTC interrupt flag
<code>CTC_INT_FLAG_CKO</code> <code>K</code>	clock trim OK interrupt
<code>CTC_INT_FLAG_CKW</code> <code>ARN</code>	clock trim warning interrupt
<code>CTC_INT_FLAG_ERR</code>	error interrupt
<code>CTC_INT_FLAG_EREF</code>	expect reference interrupt
<code>CTC_INT_FLAG_CKE</code> <code>RR</code>	clock trim error bit interrupt
<code>CTC_INT_FLAG_REF</code> <code>MISS</code>	reference sync pulse miss interrupt
<code>CTC_INT_FLAG_TRIM</code> <code>ERR</code>	trim value error interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

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

ctc_interrupt_flag_clear

The description of `ctc_interrupt_flag_clear` is shown as below:

Table 3-156. Function `ctc_interrupt_flag_clear`

Function name	ctc_interrupt_flag_clear
Function prototype	<code>void ctc_interrupt_flag_clear(uint32_t int_flag);</code>
Function descriptions	clear CTC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
<code>int_flag</code>	CTC interrupt flag
<code>CTC_INT_FLAG_CKO_K</code>	clock trim OK interrupt
<code>CTC_INT_FLAG_CKWARN</code>	clock trim warning interrupt
<code>CTC_INT_FLAG_ERR</code>	error interrupt
<code>CTC_INT_FLAG_EREF</code>	expect reference interrupt
<code>CTC_INT_FLAG_CKEERR</code>	clock trim error bit interrupt
<code>CTC_INT_FLAG_REFMISS</code>	reference sync pulse miss interrupt
<code>CTC_INT_FLAG_TRIMERR</code>	trim value error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

3.7. CMP

The general purpose comparators can work either standalone (all terminals are available on I/Os) or together with the timers. Its blanking function can be used for false overcurrent detection in motor control applications. The CMP registers are listed in chapter [3.7.1](#), the CMP firmware functions are introduced in chapter [3.7.2](#).

3.7.1. Descriptions of Peripheral registers

CMP registers are listed in the table shown as below:

Table 3-157. CMP Registers

Registers	Descriptions
CMP1_CS	CMP1 Control/status register
CMP3_CS	CMP3 Control/status register
CMP5_CS	CMP5 Control/status register

3.7.2. Descriptions of Peripheral functions

CMP registers are listed in the table shown as below:

Table 3-158. CMP firmware function

Function name	Function description
cmp_deinit	Deinit CMP unit
cmp_input_init	CMP input init
cmp_output_init	CMP output init
cmp_outputblank_init	CMP output blank init
cmp_enable	enable CMP
cmp_disable	disable CMP
cmp_lock_enable	lock the CMP
cmp_output_level_get	get output level

Enum cmp_enum

Table 3-159. Enum cmp_enum

Member name	Function description
CMP1	Cmoparator 1
CMP3	Cmoparator 3
CMP5	Cmoparator 5

Enum inverting_input_enum

Table 3-160. Enum inverting_input_enum

Member name	Function description
CMP_1_4VREFINT	VREFINT /4 input
CMP_1_2VREFINT	VREFINT /2 input
CMP_3_4VREFINT	VREFINT *3/4 input
CMP_VREFINT	VREFINT input
CMP_PA4	PA4 (DAC0) input
CMP_PA5	PA5 input
CMP_PA2	PA2 input
CMP_PB_2_15	PB2 or PB15 input

Enum cmp_output_enum

Table 3-161. Enum cmp_output_enum

Member name	Function description
CMP_OUTPUT_NONE	output no selection
CMP_OUTPUT_TIMER0_BKIN	TIMER 0 break input
CMP_OUTPUT_TIMER2IC2_TIM ER1IC1	TIMER 2 channel2 or TIMER 1 channel1 input
CMP_OUTPUT_TIMER0IC0	TIMER 0 channel0 input
CMP_OUTPUT_TIMER1IC3	TIMER 1 channel3 input
CMP_OUTPUT_TIMER2IC0	TIMER 2 channel0 input

Enum cmp_outputblank_enum

Table 3-162. Enum cmp_outputblank_enum

Member name	Function description
CMP_OUTPUTBLANK_NONE	output no blanking
CMP_OUTPUTBLANK_TIMER2_I C3	select TIMER2_CH3 as blanking source
CMP_OUTPUTBLANK_TIMER1C 2	select TIMER1_CH2 as blanking source
CMP_OUTPUTBLANK_TIMER2IC _TIMER1IC3	select TIMER2_CH2 or TIMER1_CH3 as blanking source

cmp_deinit

The description of cmp_deinit is shown as below:

Table 3-163. Function cmp_deinit

Function name	cmp_deinit
Function prototype	void cmp_deinit (void);
Function descriptions	deinitialize comparator
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize comparator */

cmp_deinit();
```

cmp_input_init

The description of cmp_input_init is shown as below:

Table 3-164. Function cmp_input_init

Function name	cmp_input_init
Function prototype	void cmp_input_init(cmp_enum cmp_periph,inverting_input_enum inverting_input);
Function descriptions	initialize comparator input
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	CMP peripherals, refer to Enum cmp_enum
CMPx	Peripheral CMPx, (x=1,3,5)
Input parameter{in}	
inverting_input	CMP inverting input select, refer to Enum inverting_input_enum
CMP_1_4VREFINT	VREFINT *1/4 input
CMP_1_2VREFINT	VREFINT *1/2 input
CMP_3_4VREFINT	VREFINT *3/4 input
CMP_VREFINT	VREFINT input
CMP_PA4	PA4 input
CMP_PA5	PA5 input
CMP_PA2	PA2 input(only CMP1)
CMP_PB_2_15	PB2 input(only CMP1),PB15 input(only CMP3)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize comparator 1 input */
cmp_input_init (CMP1, CMP_1_4VREFINT);
```

cmp_output_init

The description of cmp_output_init is shown as below:

Table 3-165. Function cmp_output_init

Function name	cmp_output_init
Function prototype	cmp_output_init(cmp_enum cmp_periph,cmp_output_enum output_selection, uint32_t output_polarity);
Function descriptions	initialize comparator output
Precondition	-
The called functions	-

Input parameter{in}	
cmp_periph	CMP peripherals, refer to Enum cmp_enum
CMPx	Peripheral CMPx, (x=1,3,5)
Input parameter{in}	
output_selection	CMP output select, refer to Enum cmp_output_enum
CMP_OUTPUT_NONE	output no selection
CMP_OUTPUT_TIMER_0_BKIN	TIMER0 break input
CMP_OUTPUT_TIMER_2IC2_TIMER1IC1	TIMER2 channel2 input capture(CMP3), TIMER1 channel1 input capture(CMP5)
CMP_OUTPUT_TIMER_0IC0	TIMER0 channel0 input capture(CMP1)
CMP_OUTPUT_TIMER_1IC3	TIMER1 channel3 input capture(CMP1)
CMP_OUTPUT_TIMER_2IC0	TIMER2 channel0 input capture(CMP1)
Input parameter{in}	
output_polarity	CMP output polarity select
CMP_OUTPUT_POLARITY_INVERTED	output is inverted
CMP_OUTPUT_POLARITY_NOINVERTED	output is not inverted
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize comparator 1 output */

cmp_output_init(CMP1, CMP_OUTPUT_NONE,
                CMP_OUTPUT_POLARITY_NOINVERTED);
```

cmp_outputblank_init

The description of cmp_outputblank_init is shown as below:

Table 3-166. Function cmp_outputblank_init

Function name	cmp_outputblank_init
Function prototype	void cmp_outputblank_init(cmp_enum cmp_periph, cmp_outputblank_enum output_blank);
Function descriptions	initialize CMP output blanking
Precondition	-
The called functions	-

Input parameter{in}	
cmp_periph	CMP peripherals, refer to Enum cmp_enum
CMPx	Peripheral CMPx, (x=1,3,5)
Input parameter{in}	
output_blank	CMP output blank select, refer to Enum cmp_outputblank_enum
CMP_OUTPUTBLANK_NONE	no blanking
CMP_OUTPUTBLANK_TIMER2_IC3	select TIMER2_CH3 as blanking source(CMP3)
CMP_OUTPUTBLANK_TIMER1IC2	select TIMER1_CH2 as blanking source(CMP1)
CMP_OUTPUTBLANK_TIMER2IC2_TIMER1IC3	Select TIMER2_CH2 as blanking source(CMP1), select TIMER1_CH3 as blanking source(CMP5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize CMP1 output blanking */
cmp_outputblank_init (CMP1, CMP_OUTPUTBLANK_TIMER1IC2);
```

cmp_enable

The description of cmp_enable is shown as below:

Table 3-167. Function cmp_enable

Function name	cmp_enable
Function prototype	void cmp_enable(cmp_enum cmp_periph);
Function descriptions	enable comparator
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	CMP peripherals, refer to Enum cmp_enum
CMPx	Peripheral CMPx, (x=1,3,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable comparator1 */
```

```
cmp_enable (CMP1);
```

cmp_disable

The description of cmp_disable is shown as below:

Table 3-168. Function cmp_disable

Function name	cmp_disable
Function prototype	void cmp_disable(cmp_enum cmp_periph);
Function descriptions	disable comparator
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	CMP peripherals, refer to Enum cmp_enum
CMPx	Peripheral CMPx, (x=1,3,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable comparator1 */

cmp_disable (CMP1);
```

cmp_lock_enable

The description of cmp_lock_enable is shown as below:

Table 3-169. Function cmp_lock_enable

Function name	cmp_lock_enable
Function prototype	void cmp_lock_enable(cmp_enum cmp_periph);
Function descriptions	lock the comparator
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	CMP peripherals, refer to Enum cmp_enum
CMPx	Peripheral CMPx, (x=1,3,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock the comparator1 */
```

```
cmp_lock_enable (CMP1);
```

cmp_output_level_get

The description of cmp_output_level_get is shown as below:

Table 3-170. Function cmp_output_level_get

Function name	cmp_output_level_get
Function prototype	uint32_t cmp_output_level_get(cmp_enum cmp_periph);
Function descriptions	get output level
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	CMP peripherals, refer to Enum cmp_enum
CMPx	Peripheral CMPx, (x=1,3,5)
Output parameter{out}	
-	-
Return value	
uint32_t	CMP_OUTPUTLEVEL_HIGH or CMP_OUTPUTLEVEL_LOW

Example:

```
/* get CMP1 output level*/
cmp_output_level_get(CMP1);
```

3.8. 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 [3.8.1](#), the DAC firmware functions are introduced in chapter [3.8.2](#).

3.8.1. Descriptions of Peripheral registers

DAC registers are listed in the table shown as below:

Table 3-171. DAC Registers

Registers	Descriptions
DAC_CTL0	DAC control register 0
DAC_SWT	DAC software trigger register
OUT0_R12DH	DAC_OUT0 12-bit right-aligned data holding register
OUT0_L12DH	DAC_OUT0 12-bit left-aligned data holding register
OUT0_R8DH	DAC_OUT0 8-bit right-aligned data holding register
OUT1_R12DH	DAC_OUT1 12-bit right-aligned data holding register
OUT1_L12DH	DAC_OUT1 12-bit left-aligned data holding register

Registers	Descriptions
OUT1_R8DH	DAC_OUT1 8-bit right-aligned data holding register
DACC_R12DH	DAC concurrent mode 12-bit right-aligned data holding register
DACC_L12DH	DAC concurrent mode 12-bit left-aligned data holding register
DACC_R8DH	DAC concurrent mode 8-bit right-aligned data holding register
OUT0_DO	DAC_OUT0 data output register
OUT1_DO	DAC_OUT1 data output register
DAC_STAT0	DAC status register 0
DAC_CTL1	DAC control register 1
DAC_STAT1	DAC status register 1

3.8.2. Descriptions of Peripheral functions

DAC registers are listed in the table shown as below:

Table 3-172. DAC firmware function

Function name	Function description
dac_deinit	deinitialize DAC
dac_enable	enable DAC
dac_disable	disable DAC
dac_dma_enable	enable DAC DMA
dac_dma_disable	disable DAC DMA
dac_output_buffer_enable	enable DAC output buffer
dac_output_buffer_disable	disable DAC output buffer
dac_output_value_get	get DAC output value
dac_data_set	set DAC data holding register value
dac_output_fifo_enable	enable DAC output FIFO
dac_output_fifo_disable	disable DAC output FIFO
dac_output_fifo_number_get	get DAC output FIFO number
dac_trigger_enable	enable DAC trigger
dac_trigger_disable	disable DAC trigger
dac_trigger_source_config	configure DAC trigger source
dac_software_trigger_enable	enable DAC software trigger
dac_software_trigger_disable	disable DAC software trigger
dac_wave_mode_config	configure DAC wave mode
dac_wave_bit_width_config	configure DAC wave bit width
dac_lfsr_noise_config	configure DAC LFSR noise mode
dac_triangle_noise_config	configure DAC triangle noise mode
dac_concurrent_enable	enable DAC concurrent mode
dac_concurrent_disable	disable DAC concurrent mode
dac_concurrent_software_trigger_enable	enable DAC concurrent software trigger
dac_concurrent_software_trigger_disable	disable DAC concurrent software trigger

Function name	Function description
_disable	
dac_concurrent_output_buffer_enable	enable DAC concurrent buffer function
dac_concurrent_output_buffer_disable	disable DAC concurrent buffer function
dac_concurrent_data_set	set DAC concurrent mode data holding register value
dac_flag_get	get the DAC flag
dac_flag_clear	clear the DAC flag
dac_interrupt_enable	enable DAC interrupt
dac_interrupt_disable	disable DAC interrupt
dac_interrupt_flag_get	get DAC interrupt flag
dac_interrupt_flag_clear	clear DAC interrupt flag

dac_deinit

The description of `dac_deinit` is shown as below:

Table 3-173. Function `dac_deinit`

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

Example:

```
/* deinitialize DAC */

dac_deinit();
```

dac_enable

The description of `dac_enable` is shown as below:

Table 3-174. Function `dac_enable`

Function name	dac_enable
Function prototype	<code>void dac_enable(uint8_t dac_out);</code>
Function descriptions	enable DAC
Precondition	-
The called functions	-

Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC_OUT0 */

dac_enable(DAC_OUT_0);
```

dac_disable

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

Table 3-175. Function **dac_disable**

Function name	dac_disable
Function prototype	void dac_disable(uint8_t dac_out);
Function descriptions	disable DAC
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC_OUT0 */

dac_disable(DAC_OUT_0);
```

dac_dma_enable

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

Table 3-176. Function **dac_dma_enable**

Function name	dac_dma_enable
Function prototype	void dac_dma_enable(uint8_t dac_out);
Function descriptions	enable DAC DMA function
Precondition	-
The called functions	-

Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC_OUT0 DMA function */

dac_dma_enable(DAC_OUT_0);
```

dac_dma_disable

The description of `dac_dma_disable` is shown as below:

Table 3-177. Function `dac_dma_disable`

Function name	dac_dma_disable
Function prototype	void dac_dma_disable(uint8_t dac_out);
Function descriptions	disable DAC DMA function
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC_OUT0 DMA function */

dac_dma_disable(DAC_OUT_0);
```

dac_output_buffer_enable

The description of `dac_output_buffer_enable` is shown as below:

Table 3-178. Function `dac_output_buffer_enable`

Function name	dac_output_buffer_enable
Function prototype	void dac_output_buffer_enable(uint8_t dac_out);
Function descriptions	enable DAC output buffer
Precondition	-
The called functions	-

Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC_OUT0 output buffer */

dac_output_buffer_enable(DAC_OUT_0);
```

dac_output_buffer_disable

The description of `dac_output_buffer_disable` is shown as below:

Table 3-179. Function `dac_output_buffer_disable`

Function name	dac_output_buffer_disable
Function prototype	void dac_output_buffer_disable(uint8_t dac_out);
Function descriptions	disable DAC output buffer
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC_OUT0 output buffer */

dac_output_buffer_disable(DAC_OUT_0);
```

dac_output_value_get

The description of `dac_output_value_get` is shown as below:

Table 3-180. Function `dac_output_value_get`

Function name	dac_output_value_get
Function prototype	uint16_t dac_output_value_get(uint8_t dac_out);
Function descriptions	get DAC output value
Precondition	-
The called functions	-

Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection (x = 0,1)
Output parameter{out}	
-	-
Return value	
uint16_t	DAC output data (0~4095)

Example:

```
/* get DAC_OUT0 output value */

data = dac_output_value_get(DAC_OUT_0);
```

dac_data_set

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

Table 3-181. Function **dac_data_set**

Function name	dac_data_set
Function prototype	void dac_data_set(uint8_t dac_out, uint32_t dac_align, uint16_t data);
Function descriptions	set the DAC specified data holding register value
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x = 0,1)
Input parameter{in}	
dac_align	DAC align mode
DAC_ALIGN_12B_R	data left 12 bit alignment
DAC_ALIGN_12B_L	data right 12 bit alignment
DAC_ALIGN_8B_R	data right 8 bit alignment
Input parameter{in}	
data	the data sending to holding register (0~4095)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the DAC_OUT0 specified data holding register value */

dac_data_set(DAC_OUT_0, DAC_ALIGN_8B_R, 0xff);
```

dac_output_fifo_enable

The description of `dac_output_fifo_enable` is shown as below:

Table 3-182. Function `dac_output_buffer_enable`

Function name	dac_output_fifo_enable
Function prototype	void dac_output_fifo_enable(uint8_t dac_out);
Function descriptions	enable DAC output FIFO
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC_OUT0 output FIFO */

dac_output_fifo_enable(DAC_OUT_0);
```

dac_output_fifo_disable

The description of `dac_output_fifo_disable` is shown as below:

Table 3-183. Function `dac_output_fifo_disable`

Function name	dac_output_fifo_disable
Function prototype	void dac_output_fifo_disable(uint8_t dac_out);
Function descriptions	disable DAC output FIFO
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC_OUT0 output FIFO */

dac_output_fifo_disable(DAC_OUT_0);
```

dac_output_fifo_number_get

The description of `dac_output_fifo_number_get` is shown as below:

Table 3-184. Function `dac_output_fifo_number_get`

Function name	dac_output_fifo_number_get
Function prototype	uint16_t dac_output_fifo_number_get(uint8_t dac_out);
Function descriptions	get DAC output FIFO number
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection (x = 0,1)
Output parameter{out}	
-	-
Return value	
uint16_t	DAC output FIFO number (0~4)

Example:

```
/* get DAC_OUT0 output FIFO number */
data = dac_output_fifo_number_get (DAC_OUT_0);
```

dac_trigger_enable

The description of `dac_trigger_enable` is shown as below:

Table 3-185. Function `dac_trigger_enable`

Function name	dac_trigger_enable
Function prototype	void dac_trigger_enable(uint8_t dac_out);
Function descriptions	enable DAC trigger
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC_OUT0 trigger */
dac_trigger_enable(DAC_OUT_0);
```

dac_trigger_disable

The description of `dac_trigger_disable` is shown as below:

Table 3-186. Function `dac_trigger_disable`

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

Example:

```
/* disable DAC_OUT0 trigger */

dac_trigger_disable(DAC_OUT_0);
```

dac_trigger_source_config

The description of `dac_trigger_source_config` is shown as below:

Table 3-187. Function `dac_trigger_source_config`

Function name	<code>dac_trigger_source_config</code>
Function prototype	<code>void dac_trigger_source_config(uint8_t dac_out,uint32_t triggersource);</code>
Function descriptions	set DAC trigger source
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x =0,1)
Input parameter{in}	
triggersource	external triggers of DAC
DAC_TRIGGER_T5_TRGO	TIMER5 TRGO
DAC_TRIGGER_T7_TRGO	TIMER7 TRGO (for GD32E50X_HD)
DAC_TRIGGER_T2_TRGO	TIMER2 TRGO (for GD32E50X_CL)
DAC_TRIGGER_T6	TIMER6 TRGO

<i>TRGO</i>	
<i>DAC_TRIGGER_T4_TRGO</i>	TIMER4 TRGO
<i>DAC_TRIGGER_T1_TRGO</i>	TIMER1 TRGO
<i>DAC_TRIGGER_T3_TRGO</i>	TIMER3 TRGO
<i>DAC_TRIGGER_EXTI_9</i>	EXTI interrupt line 9 event
<i>DAC_TRIGGER_SOFTWARE</i>	software trigger
<i>DAC_TRIGGER_SHRTIMER_DACTRIG0</i>	SHRTIMER_DACTRIG0 trigger (for GD32E50X_HD and GD32E50X_CL devices)
<i>DAC_TRIGGER_SHRTIMER_DACTRIG1</i>	SHRTIMER_DACTRIG1 trigger (for GD32E50X_HD and GD32E50X_CL devices)
<i>DAC_TRIGGER_SHRTIMER_DACTRIG2</i>	SHRTIMER_DACTRIG2 trigger (for GD32E50X_HD and GD32E50X_CL devices)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DAC_OUT0 trigger source*/
dac_trigger_source_config(DAC_OUT_0, DAC_TRIGGER_T1_TRGO);
```

dac_software_trigger_enable

The description of `dac_software_trigger_enable` is shown as below:

Table 3-188. Function `dac_software_trigger_enable`

Function name	<code>dac_software_trigger_enable</code>
Function prototype	<code>void dac_software_trigger_enable(uint8_t dac_out);</code>
Function descriptions	enable DAC software trigger
Precondition	-
The called functions	-
Input parameter{in}	
<code>dac_out</code>	DAC output
<code>DAC_OUT_x</code>	DAC output selection(x =0,1)
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable DAC_OUT0 software trigger */
dac_software_trigger_enable(DAC_OUT_0);
```

dac_software_trigger_disable

The description of `dac_software_trigger_disable` is shown as below:

Table 3-189. Function `dac_software_trigger_disable`

Function name	dac_software_trigger_disable
Function prototype	void dac_software_trigger_disable(uint8_t dac_out);
Function descriptions	disable DAC software trigger
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x =0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC_OUT0 software trigger */
dac_software_trigger_disable(DAC_OUT_0);
```

dac_wave_mode_config

The description of `dac_wave_mode_config` is shown as below:

Table 3-190. Function `dac_wave_mode_config`

Function name	dac_wave_mode_config
Function prototype	void dac_wave_mode_config(uint8_t dac_out, uint32_t wave_mode);
Function descriptions	configure DAC wave mode
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x =0,1)
Input parameter{in}	

wave_mode	wave mode
DAC_WAVE_DISABLE	wave disable
DAC_WAVE_MODE_LFSR	LFSR noise mode
DAC_WAVE_MODE_TRIANGLE	triangle noise mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DAC_OUT0 wave mode */
dac_wave_mode_config(DAC_OUT_0, DAC_WAVE_DISABLE);
```

dac_wave_bit_width_config

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

Table 3-191. Function **dac_wave_bit_width_config**

Function name	dac_wave_bit_width_config
Function prototype	void dac_wave_bit_width_config(uint8_t dac_out, uint32_t bit_width);
Function descriptions	configure DAC wave bit width
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x =0,1)
Input parameter{in}	
bit_width	noise wave bit width
DAC_WAVE_BIT_WIDTH_x	noise wave bit width (x = 1..12)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DAC_OUT0 wave bit width */
dac_wave_bit_width_config(DAC_OUT_0, DAC_WAVE_BIT_WIDTH_1);
```

dac_lfsr_noise_config

The description of `dac_lfsr_noise_config` is shown as below:

Table 3-192. Function `dac_lfsr_noise_config`

Function name	dac_lfsr_noise_config
Function prototype	void dac_lfsr_noise_config(uint8_t dac_out, uint32_t unmask_bits);
Function descriptions	configure DAC LFSR noise mode
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x =0,1)
Input parameter{in}	
unmask_bits	noise wave unmask bit width
DAC_LFSR_BIT0	unmask the LFSR bit 0
DAC_LFSR_BITSx_0	unmask the LFSR bits[x:0]
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DAC_OUT0 LFSR noise mode */

dac_lfsr_noise_config(DAC_OUT_0, DAC_LFSR_BIT0);
```

dac_triangle_noise_config

The description of `dac_triangle_noise_config` is shown as below:

Table 3-193. Function `dac_triangle_noise_config`

Function name	dac_triangle_noise_config
Function prototype	void dac_triangle_noise_config(uint8_t dac_out, uint32_t amplitude);
Function descriptions	configure DAC triangle noise mode
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x =0,1)
Input parameter{in}	
amplitude	the amplitude of triangle noise
DAC_TRIANGLE_AMPLITUDE_x	$x = 2^n-1$ ($n = 1..12$)
Output parameter{out}	

-	-	-
Return value		
-	-	-

Example:

```
/* configure DAC_OUT0 triangle noise mode */
dac_triangle_noise_config(DAC_OUT_0, DAC_TRIANGLE_AMPLITUDE_1);
```

dac_concurrent_enable

The description of `dac_concurrent_enable` is shown as below:

Table 3-194. Function `dac_concurrent_enable`

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

Example:

```
/* enable DAC concurrent mode */
dac_concurrent_enable();
```

dac_concurrent_disable

The description of `dac_concurrent_disable` is shown as below:

Table 3-195. Function `dac_concurrent_disable`

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

-	-	-
---	---	---

Example:

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

```
dac_concurrent_disable();
```

dac_concurrent_software_trigger_enable

The description of `dac_concurrent_software_trigger_enable` is shown as below:

Table 3-196. Function `dac_concurrent_software_trigger_enable`

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

Example:

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

```
dac_concurrent_software_trigger_enable();
```

dac_concurrent_software_trigger_disable

The description of `dac_concurrent_software_trigger_disable` is shown as below:

Table 3-197. Function `dac_concurrent_software_trigger_disable`

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

Example:

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

dac_concurrent_software_trigger_disable();
```

dac_concurrent_output_buffer_enable

The description of `dac_concurrent_output_buffer_enable` is shown as below:

Table 3-198. Function `dac_concurrent_output_buffer_enable`

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

Example:

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

dac_concurrent_output_buffer_enable();
```

dac_concurrent_output_buffer_disable

The description of `dac_concurrent_output_buffer_disable` is shown as below:

Table 3-199. Function `dac_concurrent_output_buffer_disable`

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

Example:

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

```
dac_concurrent_output_buffer_disable();
```

dac_concurrent_data_set

The description of `dac_concurrent_data_set` is shown as below:

Table 3-200. Function `dac_concurrent_data_set`

Function name	dac_concurrent_data_set
Function prototype	void dac_concurrent_data_set(uint32_t dac_align, uint16_t data0, uint16_t data1);
Function descriptions	set DAC concurrent mode data holding register value
Precondition	-
The called functions	-
Input parameter{in}	
dac_align	DAC align mode
<i>DAC_ALIGN_12B_R</i>	data left 12 bit alignment
<i>DAC_ALIGN_12B_L</i>	data right 12 bit alignment
<i>DAC_ALIGN_8B_R</i>	data right 8 bit alignment
Input parameter{in}	
data0	the data sending to holding register of DAC_OUT0(0~4095)
Input parameter{in}	
data1	the data sending to holding register of DAC_OUT1(0~4095)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

dac_flag_get

The description of `dac_flag_get` is shown as below:

Table 3-201. Function `dac_flag_get`

Function name	dac_flag_get
Function prototype	FlagStatus dac_flag_get(uint8_t dac_out, uint32_t flag);
Function descriptions	get DAC flag
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output

<i>DAC_OUT_x</i>	DAC output selection(x =0,1)
Input parameter{in}	
<i>flag</i>	DAC flag
<i>DAC_FLAG_DDUDR0</i>	DAC_OUT0 DMA underrun flag
<i>DAC_FLAG_FF0</i>	DAC_OUT0 FIFO full flag
<i>DAC_FLAG_FE0</i>	DAC_OUT0 FIFO empty flag
<i>DAC_FLAG_FIFOOVR0</i>	DAC_OUT0 FIFO overflow flag
<i>DAC_FLAG_FIFOUDR0</i>	DAC_OUT0 FIFO underflow flag
<i>DAC_FLAG_DDUDR1</i>	DAC_OUT1 DMA underrun flag
<i>DAC_FLAG_FF1</i>	DAC_OUT1 FIFO full flag
<i>DAC_FLAG_FE1</i>	DAC_OUT1 FIFO empty flag
<i>DAC_FLAG_FIFOOVR1</i>	DAC_OUT1 FIFO overflow flag
<i>DAC_FLAG_FIFOUDR1</i>	DAC_OUT1 FIFO underflow flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the DAC flag */

FlagStatus flag_value;

flag_value = dac_flag_get(DAC_OUT_0, DAC_FLAG_FIFOOVR0);
```

dac_flag_clear

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

Table 3-202. Function **dac_flag_clear**

Function name	dac_flag_clear
Function prototype	void dac_flag_clear(uint8_t dac_out, uint32_t flag);
Function descriptions	clear DAC flag
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUT_x</i>	DAC output selection(x =0,1)
Input parameter{in}	
<i>flag</i>	DAC flag
<i>DAC_FLAG_DDUDR0</i>	DAC_OUT0 DMA underrun flag

<i>DAC_FLAG_FIFOOVR0</i>	DAC_OUT0 FIFO overflow flag
<i>DAC_FLAG_FIFOUDR0</i>	DAC_OUT0 FIFO underflow flag
<i>DAC_FLAG_DDUDR1</i>	DAC_OUT1 DMA underrun flag
<i>DAC_FLAG_FIFOOVR1</i>	DAC_OUT1 FIFO overflow flag
<i>DAC_FLAG_FIFOUDR1</i>	DAC_OUT1 FIFO underflow flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the DAC flag */
dac_flag_clear(DAC_OUT_0, DAC_FLAG_FIFOOVR0);
```

dac_interrupt_enable

The description of `dac_interrupt_enable` is shown as below:

Table 3-203. Function `dac_interrupt_enable`

Function name	<code>dac_interrupt_enable</code>
Function prototype	<code>void dac_interrupt_enable(uint8_t dac_out, uint32_t interrupt);</code>
Function descriptions	enable DAC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
<code>dac_out</code>	DAC output
<code>DAC_OUT_x</code>	DAC output selection(x =0,1)
Input parameter{in}	
<code>interrupt</code>	DAC interrupt
<code>DAC_INT_DDUDRIE0</code>	DAC_OUT0 DMA underrun interrupt enable
<code>DAC_INT_FIFOOVRIE0</code>	DAC_OUT0 FIFO overflow interrupt enable
<code>DAC_INT_FIFOUDRIE0</code>	DAC_OUT0 FIFO underflow interrupt enable
<code>DAC_INT_DDUDRIE1</code>	DAC_OUT1 DMA underrun interrupt enable
<code>DAC_INT_FIFOOVRIE1</code>	DAC_OUT1 FIFO overflow interrupt enable
<code>DAC_INT_FIFOUDRIE1</code>	DAC_OUT1 FIFO underflow interrupt enable

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC interrupt */
dac_interrupt_enable(DAC_OUT_0, DAC_INT_DDUDRIE0);
```

dac_interrupt_disable

The description of `dac_interrupt_disable` is shown as below:

Table 3-204. Function `dac_interrupt_enable`

Function name	<code>dac_interrupt_enable</code>
Function prototype	<code>void dac_interrupt_disable(uint8_t dac_out, uint32_t interrupt);</code>
Function descriptions	disable DAC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
DAC_OUT_x	DAC output selection(x =0,1)
Input parameter{in}	
interrupt	DAC interrupt
DAC_INT_DDUDRIE0	DAC_OUT0 DMA underrun interrupt disable
DAC_INT_FIFOOVRIE0	DAC_OUT0 FIFO overflow interrupt disable
DAC_INT_FIFOUDRIE0	DAC_OUT0 FIFO underflow interrupt disable
DAC_INT_DDUDRIE1	DAC_OUT1 DMA underrun interrupt disable
DAC_INT_FIFOOVRIE1	DAC_OUT1 FIFO overflow interrupt disable
DAC_INT_FIFOUDRIE1	DAC_OUT1 FIFO underflow interrupt disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC interrupt */
dac_interrupt_disable(DAC_OUT_0, DAC_INT_DDUDRIE0);
```

dac_interrupt_flag_get

The description of `dac_interrupt_flag_get` is shown as below:

Table 3-205. Function `dac_interrupt_flag_get`

Function name	<code>dac_interrupt_flag_get</code>
Function prototype	<code>FlagStatus dac_interrupt_flag_get(uint8_t dac_out, uint32_t int_flag);</code>
Function descriptions	get DAC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
<code>dac_out</code>	DAC output
<code>DAC_OUT_x</code>	DAC output selection(x =0,1)
Input parameter{in}	
<code>int_flag</code>	DAC interrupt flag
<code>DAC_INT_FLAG_DDUDR0</code>	DAC_OUT0 DMA underrun interrupt flag
<code>DAC_INT_FLAG_FIFOVR0</code>	DAC_OUT0 FIFO overflow interrupt flag
<code>DAC_INT_FLAG_FIFOUDR0</code>	DAC_OUT0 FIFO underflow interrupt flag
<code>DAC_INT_FLAG_DDUDR1</code>	DAC_OUT1 DMA underrun interrupt flag
<code>DAC_INT_FLAG_FIFOVR1</code>	DAC_OUT1 FIFO overflow interrupt flag
<code>DAC_INT_FLAG_FIFOUDR1</code>	DAC_OUT1 FIFO underflow interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the DAC interrupt flag */

FlagStatus flag_value;

flag_value = dac_interrupt_flag_get(DAC_OUT_0, DAC_INT_FLAG_FIFOVR0);
```

dac_interrupt_flag_clear

The description of `dac_interrupt_flag_clear` is shown as below:

Table 3-206. Function `dac_interrupt_flag_clear`

Function name	<code>dac_interrupt_flag_clear</code>
----------------------	---------------------------------------

Function prototype	void dac_interrupt_flag_clear(uint8_t dac_out, uint32_t int_flag);
Function descriptions	clear DAC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUT_x</i>	DAC output selection(x =0,1)
Input parameter{in}	
int_flag	DAC interrupt flag
<i>DAC_INT_FLAG_DDUDR0</i>	DAC_OUT0 DMA underrun interrupt flag
<i>DAC_INT_FLAG_FIFOVR0</i>	DAC_OUT0 FIFO overflow interrupt flag
<i>DAC_INT_FLAG_FIFOUDR0</i>	DAC_OUT0 FIFO underflow interrupt flag
<i>DAC_INT_FLAG_DDUDR1</i>	DAC_OUT1 DMA underrun interrupt flag
<i>DAC_INT_FLAG_FIFOVR1</i>	DAC_OUT1 FIFO overflow interrupt flag
<i>DAC_INT_FLAG_FIFOUDR1</i>	DAC_OUT1 FIFO underflow interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* clear DAC interrupt flag */

dac_interrupt_flag_clear (DAC_OUT_0, DAC_INT_FLAG_FIFOVR0);
```

3.9. DBG

The DBG hold unit helps debugger to debug power saving mode. The DBG registers are listed in chapter [3.9.1](#), the DBG firmware functions are introduced in chapter [3.9.2](#).

3.9.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

Table 3-207. DBG Registers

Registers	Descriptions
<i>DBG_ID</i>	DBG ID code register

Registers	Descriptions
DBG_CTL	DBG control register

3.9.2. Descriptions of Peripheral functions

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

Table 3-208. DBG firmware function

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

Enum `dbg_periph_enum`

Table 3-209. Enum `dbg_periph_enum`

Member name	Function description
DBG_CAN2_HOLD	hold CAN2 receive register counter when core is halted
DBG_FWDGT_HOLD	hold FWDGT counter when core is halted
DBG_WWDGT_HOLD	hold WWDGT counter 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	hold CAN0 receive register counter 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	hold CAN1 receive register counter when core
DBG_I2C2_HOLD	hold I2C2 smbus timeout 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_SHRTIMER_HOLD	hold SHRTIMER counter when core is halted, except for GD32EPRT series

dbg_deinit

The description of dbg_deinit is shown as below:

Table 3-210. Function dbg_deinit

Function name	dbg_deinit
Function prototype	void dbg_deinit(void);
Function descriptions	deinitialize the DBG
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize the DBG */

dbg_deinit();
```

dbg_id_get

The description of dbg_id_get is shown as below:

Table 3-211. Function dbg_id_get

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

Example:

```
/* read DBG_ID code register */
```

```
uint32_t id_value = 0;
```

```
id_value = dbg_id_get();
```

dbg_low_power_enable

The description of `dbg_low_power_enable` is shown as below:

Table 3-212. 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}	
<code>dbg_low_power</code>	low power mode
<code>DBG_LOW_POWER_SLEEP</code>	keep debugger connection during sleep mode
<code>DBG_LOW_POWER_DEEPSLEEP</code>	keep debugger connection during deepsleep mode
<code>DBG_LOW_POWER_STANDBY</code>	keep debugger connection during standby mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* keep debugger connection during sleep 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 3-213. 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}	
<code>dbg_low_power</code>	low power mode

<i>DBG_LOW_POWER_SLEEP</i>	keep debugger connection during sleep mode
<i>DBG_LOW_POWER_DEEPSLEEP</i>	keep debugger connection during deepsleep mode
<i>DBG_LOW_POWER_STANDBY</i>	keep debugger connection during standby mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* do not keep debugger connection during sleep mode */
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

dbg_periph_enable

The description of `dbg_periph_enable` is shown as below:

Table 3-214. 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}	
<i>dbg_periph</i>	Peripheral refer to Table 3-209. Enum <code>dbg_periph_enum</code>
<i>DBG_FWDGT_HOLD</i>	debug FWDGT kept when core is halted
<i>DBG_CANx_HOLD</i>	x=0,1,2 hold CANx counter when core is halted
<i>DBG_WWDGT_HOLD</i>	debug WWDGT kept when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1,2 hold I2Cx smbus when core is halted
<i>DBG_TIMERx_HOLD</i>	x=0,1,2,3,4,5,6,7,8,9,10,11,12,13, hold TIMERx counter when core is halted. TIMER8..13 are not available for GD32EPRT series.
<i>DBG_SHRTIMER_HOLD</i>	hold SHRTIMER counter when core is halted, except for GD32EPRT series
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* hold TIMER0 counter when core is halted */
dbg_periph_enable(DBG_TIMER0_HOLD);
```

dbg_periph_disable

The description of dbg_periph_disable is shown as below:

Table 3-215. 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 3-209. 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>	x=0,1,2 hold CANx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1,2 hold I2Cx smbus when core is halted
<i>DBG_TIMERx_HOLD</i>	x=0,1,2,3,4,5,6,7,8,9,10,11,12,13, hold TIMERx counter when core is halted. TIMER8..13 are not available for GD32EPRT series.
<i>DBG_SHRTIMER_HOLD</i>	hold SHRTIMER counter when core is halted, except for GD32EPRT series
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* do not hold TIMER0 counter when core is halted */
```

```
dbg_periph_disable(DBG_TIMER0_HOLD);
```

dbg_trace_pin_enable

The description of dbg_trace_pin_enable is shown as below:

Table 3-216. 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 3-217. Function dbg_trace_pin_disable

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

Example:

```
/* disable trace pin assignment */
```

```
dbg_trace_pin_disable();
```

3.10. 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 3-218. DMA Registers

Registers	Descriptions
DMA_INTF	Interrupt flag register
DMA_INTC	Interrupt flag clear register

Registers	Descriptions
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

3.10.2. Descriptions of Peripheral functions

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

Table 3-219. 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
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 increase algorithm of memory
dma_memory_increase_disable	disable next address increase algorithm of memory
dma_periph_increase_enable	enable next address increase algorithm of peripheral
dma_periph_increase_disable	disable next address increase 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 a DMA channel flag
dma_interrupt_enable	enable DMA interrupt
dma_interrupt_disable	disable DMA interrupt

Function name	Function description
dma_interrupt_flag_get	check DMA flag and interrupt enable bit is set or not
dma_interrupt_flag_clear	clear a DMA channel flag

Enum dma_channel_enum

Table 3-220. Enum dma_channel_enum

Member name	Function description
DMA_CH0	DMA Channel0
DMA_CH1	DMA Channel1
DMA_CH2	DMA Channel2
DMA_CH3	DMA Channel3
DMA_CH4	DMA Channel4
DMA_CH5	DMA Channel5
DMA_CH6	DMA Channel6

Structure dma_parameter_struct

Table 3-221. 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 3-222. 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, refer to Table 3-220. Enum dma_channel_enum
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

dma_struct_para_init

The description of `dma_struct_para_init` is shown as below:

Table 3-223. 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}	
<i>*init_struct</i>	A <code>dma_parameter_struct</code> address, refer to Table 3-221. Structure <code>dma_parameter_struct</code>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the parameters of DMA */
dma_parameter_struct dma_init_struct;
dma_struct_para_init(&dma_init_struct);
```

dma_init

The description of `dma_init` is shown as below:

Table 3-224. Function `dma_init`

Function name	dma_init
Function prototype	void dma_init(uint32_t dma_periph, dma_channel_enum channelx, dma_parameter_struct* init_struct);
Function descriptions	initialize 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, refer to Table 3-220. Enum dma_channel enum
DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)		DMA channel selection
Input parameter{in}		
init_struct		Structure for initialization, the structure members can refer to Table 3-221. Structure dma_parameter_struct
Output parameter{out}		
-		-
Return value		
-		-

Example:

```

/* DMA0 channel0 initialize */
dma_parameter_struct dma_init_struct;
dma_deinit(DMA0, DMA_CH0);
dma_struct_para_init(&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_32BIT;
dma_init_struct.number = TRANSFER_NUM/4;
dma_init_struct.periph_addr = (uint32_t)FLASH_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_32BIT;
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 3-225. 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..4)</i>	DMA channel selection, refer to Table 3-220. Enum dma_channel enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA0 channel0 circulation mode */
dma_circulation_enable(DMA0, DMA_CH0);
```

dma_circulation_disable

The description of `dma_circulation_disable` is shown as below:

Table 3-226. 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..4)</i>	DMA channel selection, refer to Table 3-220. Enum dma_channel enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

dma_memory_to_memory_enable

The description of `dma_memory_to_memory_enable` is shown as below:

Table 3-227. 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..4)</i>	DMA channel selection, refer to Table 3-220. Enum <code>dma_channel_enum</code>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA0 channel0 memory to memory mode */
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 3-228. 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..4)</i>	DMA channel selection, refer to Table 3-220. Enum <code>dma_channel_enum</code>

6; DMA1: x=0..4)	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA0 channel0 memory to memory mode */
dma_memory_to_memory_enable(DMA0, DMA_CH0);
```

dma_channel_enable

The description of `dma_channel_enable` is shown as below:

Table 3-229. 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
DMAx(x=0,1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)	DMA channel selection, refer to Table 3-220. Enum <code>dma_channel_enum</code>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA0 channel0 */
dma_channel_enable(DMA0, DMA_CH0);
```

dma_channel_disable

The description of `dma_channel_disable` is shown as below:

Table 3-230. 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
<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..4)</i>	DMA channel selection, refer to Table 3-220. Enum dma_channel enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA0 channel0 */
dma_channel_disable(DMA0, DMA_CH0);
```

dma_periph_address_config

The description of `dma_periph_address_config` is shown as below:

Table 3-231. Function `dma_periph_address_config`

Function name	<code>dma_periph_address_config</code>
Function prototype	<code>void dma_periph_address_config(uint32_t dma_periph, 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..4)</i>	DMA channel selection, refer to Table 3-220. Enum dma_channel enum
Input parameter{in}	
address	peripheral base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

dma_memory_address_config

The description of `dma_memory_address_config` is shown as below:

Table 3-232. Function `dma_memory_address_config`

Function name	dma_memory_address_config
Function prototype	void dma_memory_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
Function descriptions	set DMA memory base address
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..4)	DMA channel selection, refer to Table 3-220. Enum <code>dma_channel_enum</code>
Input parameter{in}	
address	memory base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
uint8_t g_destbuf[TRANSFER_NUM];

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

dma_transfer_number_config

The description of `dma_transfer_number_config` is shown as below:

Table 3-233. 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..4)</i>		DMA channel selection, refer to Table 3-220. Enum dma_channel_enum
Input parameter{in}		
number		data transfer number (0x00000000 – 0x0000FFFF)
Output parameter{out}		
-		-
Return value		
-		-

Example:

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

dma_transfer_number_get
```

The description of `dma_transfer_number_get` is shown as below:

Table 3-234. Function `dma_transfer_number_get`

Function name		<code>dma_transfer_number_get</code>
Function prototype		uint32_t <code>dma_transfer_number_get(uint32_t dma_periph,</code> <code>dma_channel_enum channelx);</code>
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..4)</i>		DMA channel selection, refer to Table 3-220. Enum dma_channel_enum
Output parameter{out}		
-		-
Return value		
<code>uint32_t</code>		0x00000000 – 0x0000FFFF

Example:

```

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

```

dma_priority_config

The description of `dma_priority_config` is shown as below:

Table 3-235. 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..4)</i>	DMA channel selection, refer to Table 3-220. Enum <code>dma_channel_enum</code>
Input parameter{in}	
priority	priority Level of this channel
<i>DMA_PRIORITY_LOW</i>	low priority
<i>DMA_PRIORITY_MEDIUM</i>	medium priority
<i>DMA_PRIORITY_HIGH</i>	high priority
<i>DMA_PRIORITY_ULTRA_HIGH</i>	ultra high priority
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

dma_memory_width_config

The description of `dma_memory_width_config` is shown as below:

Table 3-236. 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
DMAx(x=0,1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)	DMA channel selection, refer to Table 3-220. Enum dma_channel enum
Input parameter{in}	
mwidth	transfer data width of memory
DMA_MEMORY_WIDT_H_8BIT	transfer data width of memory is 8-bit
DMA_MEMORY_WIDT_H_16BIT	transfer data width of memory is 16-bit
DMA_MEMORY_WIDT_H_32BIT	transfer data width of memory is 32-bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

dma_periph_width_config

The description of `dma_periph_width_config` is shown as below:

Table 3-237. 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
DMAx(x=0,1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)	DMA channel selection, refer to Table 3-220. Enum dma_channel enum

6; DMA1: x=0..4)	
Input parameter{in}	
ewidth	transfer data width of peripheral
<i>DMA_PERIPHERAL_W_IDTH_8BIT</i>	transfer data width of peripheral is 8-bit
<i>DMA_PERIPHERAL_W_IDTH_16BIT</i>	transfer data width of peripheral is 16-bit
<i>DMA_PERIPHERAL_W_IDTH_32BIT</i>	transfer data width of peripheral is 32-bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
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 3-238. 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..4)</i>	DMA channel selection, refer to Table 3-220. Enum <code>dma_channel_enum</code>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

dma_memory_increase_disable

The description of `dma_memory_increase_disable` is shown as below:

Table 3-239. 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..4)</i>	DMA channel selection, refer to Table 3-220. Enum <code>dma_channel_enum</code>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_memory_increase_disable(DMA0, DMA_CH0);
```

dma_periph_increase_enable

The description of `dma_periph_increase_enable` is shown as below:

Table 3-240. 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..4)</i>	DMA channel selection, refer to Table 3-220. Enum <code>dma_channel_enum</code>

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_periph_increase_enable(DMA0, DMA_CH0);
```

dma_periph_increase_disable

The description of `dma_periph_increase_disable` is shown as below:

Table 3-241. Function `dma_periph_increase_disable`

Function name	<code>dma_periph_increase_disable</code>
Function prototype	<code>void dma_periph_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);</code>
Function descriptions	disable next address increasement algorithm of peripheral
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..4)	DMA channel selection, refer to Table 3-220. Enum <code>dma_channel_enum</code>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

dma_transfer_direction_config

The description of `dma_transfer_direction_config` is shown as below:

Table 3-242. Function `dma_transfer_direction_config`

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

Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0..1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection, refer to Table 3-220. Enum dma_channel enum
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:

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

dma_flag_get

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

Table 3-243. 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..4)</i>	DMA channel selection, refer to Table 3-220. Enum dma_channel enum
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

DMA_FLAG_ERR	error flag of channel
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
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 3-244. 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 the flag of a 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..4)	DMA channel selection, refer to Table 3-220. Enum <code>dma_channel_enum</code>
Input parameter{in}	
flag	specify get which flag
DMA_FLAG_G	global interrupt flag of channel
DMA_FLAG_FTF	full transfer finish flag of channel
DMA_FLAG_HTF	half transfer finish flag of channel
DMA_FLAG_ERR	error flag of channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

dma_interrupt_enable

The description of `dma_interrupt_enable` is shown as below:

Table 3-245. 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..4)</i>	DMA channel selection, refer to Table 3-220. Enum <code>dma_channel_enum</code>
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 3-246. 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..4)</i>	DMA channel selection, refer to Table 3-220. Enum dma_channel enum
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 3-247. 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..4)</i>	DMA channel selection, refer to Table 3-220. Enum dma_channel enum
Input parameter{in}	
flag	specify get which flag
<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:

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

dma_interrupt_flag_clear

The description of `dma_interrupt_flag_clear` is shown as below:

Table 3-248. 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 the interrupt flag of a 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..4)	DMA channel selection, refer to Table 3-220. Enum <code>dma_channel_enum</code>
Input parameter{in}	
flag	specify get which flag
DMA_INT_FLAG_G	global interrupt flag of channel
DMA_INT_FLAG_FTF	full transfer finish interrupt flag of channel
DMA_INT_FLAG_HTF	half transfer finish interrupt flag of channel
DMA_INT_FLAG_ERR	error interrupt flag of channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_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 3-249. ENET Registers

Registers	Descriptions
ENET_MAC_CFG	MAC configuration register
ENET_MAC_FRMF	MAC frame filter register
ENET_MAC_HLH	MAC hash list high register
ENET_MAC_HLL	MAC hash list low register
ENET_MAC_PHY_CTL	MAC PHY control register
ENET_MAC_PHY_DATA	MAC PHY data register
ENET_MAC_FCTL	MAC flow control register
ENET_MAC_VLT	MAC VLAN tag register
ENET_MAC_RWFF	MAC remote wakeup frame filter register
ENET_MAC_WUM	MAC wakeup management register
ENET_MAC_DBG	MAC debug register
ENET_MAC_INTF	MAC interrupt flag register
ENET_MAC_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

Registers	Descriptions
ENET_MAC_ADDR 3H	MAC address 3 high register
ENET_MAC_ADDR 3L	MAC address 3 low register
ENET_MAC_FCTH	MAC flow control threshold register
ENET_MSC_CTL	MSC control register
ENET_MSC_RINTF	MSC receive interrupt flag register
ENET_MSC_TINTF	MSC transmit interrupt flag register
ENET_MSC_RINT MSK	MSC receive interrupt mask register
ENET_MSC_TINTM SK	MSC transmit interrupt mask register
ENET_MSC_SCCN T	MSC transmitted good frames after a single collision counter register
ENET_MSC_MSCC NT	MSC transmitted good frames after more than a single collision counter register
ENET_MSC_TGFC NT	MSC transmitted good frames counter register
ENET_MSC_RFCE CNT	MSC received frames with CRC error counter register
ENET_MSC_RFAE CNT	MSC received frames with alignment error counter register
ENET_MSC_RGUF CNT	MSC received good unicast frames counter register
ENET_PTP_TSCTL	PTP time stamp control register
ENET_PTP_SSINC	PTP subsecond increment register
ENET_PTP_TSH	PTP time stamp high register
ENET_PTP_TSL	PTP time stamp low register
ENET_PTP_TSUH	PTP time stamp update high register
ENET_PTP_TSUL	PTP time stamp update low register
ENET_PTP_TSADD END	PTP time stamp addend register
ENET_PTP_ETH	PTP expected time high register
ENET_PTP_ETL	PTP expected time low register
ENET_PTP_TSFLAG	PTP time stamp flag register
ENET_PTP_PPSCTL L	PTP PPS control register
ENET_DMA_BCTL	DMA bus control register
ENET_DMA_TPEN	DMA transmit poll enable register
ENET_DMA_RPEN	DMA receive poll enable register
ENET_DMA_RDTA DDR	DMA receive descriptor table address register

Registers	Descriptions
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_RSWD C	DMA receive state watchdog counter register
ENET_DMA_CTDA DDR	DMA current transmit descriptor address register
ENET_DMA_CRDA DDR	DMA current receive descriptor address register
ENET_DMA_CTBA 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 3-250. 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)

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

Function name	Function description
enet_rxprocess_check_recovery	check and recover the Rx process
enet_txfifo_flush	flush the ENET transmit fifo, and wait until the flush operation completes
enet_current_desc_address_get	get the transmit/receive address of current descriptor, or current buffer, or descriptor table
enet_desc_information_get	get the Tx or Rx descriptor information
enet_missed_frame_counter_get	get the number of missed frames during receiving descriptor function
enet_desc_flag_get	get the bit flag of ENET dma descriptor
enet_desc_flag_set	set the bit flag of ENET dma tx descriptor
enet_desc_flag_clear	clear the bit flag of ENET dma tx descriptor
enet_rx_desc_immediate_receive_complete_interrupt	when receiving the completed, set RS bit in ENET_DMA_STAT register will immediately set
enet_rx_desc_delay_receive_complete_interrupt	when receiving the completed, set RS bit in ENET_DMA_STAT register will be set after a configurable delay time
enet_rxframe_drop	drop current receive frame
enet_dma_feature_enable	enable DMA feature
enet_dma_feature_disable	disable DMA feature
enhanced descriptor	
enet_rx_desc_enhanced_status_get	get the bit of extended status flag in ENET DMA descriptor
enet_desc_select_enhanced_mode	configure descriptor to work in enhanced mode
enet_ptp_enhanced_descriptors_chain_init	initialize the dma Tx/Rx descriptors's parameters in enhanced chain mode with ptp function
enet_ptp_enhanced_descriptors_ring_init	initialize the dma Tx/Rx descriptors's parameters in enhanced ring mode with ptp function
enet_ptpframe_receive_enhanced_mode	receive a packet data with timestamp values to application buffer, when the DMA is in enhanced mode
enet_ptpframe_transmit_enhanced_mode	send data with timestamp values in application buffer as a transmit packet, when the DMA is in enhanced mode
normal descriptor	
enet_desc_select_normal_mode	configure descriptor to work in normal mode
enet_ptp_normal_descriptors_chain_init	initialize the dma Tx/Rx descriptors's parameters in normal chain mode with ptp function
enet_ptp_normal_descriptors_ring_init	initialize the dma Tx/Rx descriptors's parameters in normal ring mode with ptp function
enet_ptpframe_receive_normal_mode	receive a packet data with timestamp values to application buffer, when the DMA is in normal mode
enet_ptpframe_transmit_normal_mode	send data with timestamp values in application buffer as a transmit packet, when the DMA is in normal mode
WUM function	
enet_wum_filter_register_pointer_res	wakeup frame filter register pointer reset

Function name	Function description
et	
enet_wum_filter_config	set the remote wakeup frame registers
enet_wum_feature_enable	enable wakeup management features
enet_wum_feature_disable	disable wakeup management features
MSC function	
enet_msc_counters_reset	reset the MAC statistics counters
enet_msc_feature_enable	enable the MAC statistics counter features
enet_msc_feature_disable	disable the MAC statistics counter features
enet_msc_counters_preset_config	configure MAC statistics counters preset mode
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_pps_output_frequency_config	configure the PPS output frequency
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
enet_ptp_finecorrection_settime	set system time in fine method
enet_ptp_flag_get	get the ptpt flag status
internal function	
enet_initpara_reset	reset the ENET initpara struct, call it before using enet_initpara_config()

Structure enet_initpara_struct

Table 3-251. Structure enet_initpara_struct

member name	Function description
option_enable	select which function to configure
forward_frame	frame forward related parameters
dmabus_mode	DMA bus mode related parameters

dma_maxburst	DMA max burst related parameters
dma_arbitration	DMA Tx and Rx arbitration related parameters
store_forward_mode	store forward mode related parameters
dma_function	DMA control related parameters
vlan_config	VLAN tag related parameters
flow_control	flow control related parameters
hashtable_high	hash list high 32-bit related parameters
hashtable_low	hash list low 32-bit related parameters
framesfilter_mode	frame filter control related parameters
halfduplex_param	halfduplex related parameters
timer_config	frame timer related parameters
interframegap	inter frame gap related parameters

Structure enet_descriptors_struct

Table 3-252. Structure enet_descriptors_struct

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

Structure enet_ptp_systime_struct

Table 3-253. 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 3-254. 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_	flow control status flag

FLOWCONTROL	
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_TMST	timestamp trigger status flag
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_ER	early receive status flag
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_E_RROR	error during write/read transfer flag
ENET_DMA_FLAG_EB_ACCESS_ERR_OR	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 3-255. 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
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 3-256. Enum enet_int_enum

member name	Function description
ENET_MAC_INT_WUMIM	WUM interrupt mask
ENET_MAC_INT_TMSTIM	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_TE	transmit interrupt enable

<i>ENET_DMA_INT_T PSIE</i>	transmit process stopped interrupt enable
<i>ENET_DMA_INT_T BUIE</i>	transmit buffer unavailable interrupt enable
<i>ENET_DMA_INT_T JTIE</i>	transmit jabber timeout interrupt enable
<i>ENET_DMA_INT_R OIE</i>	receive overflow interrupt enable
<i>ENET_DMA_INT_T UIE</i>	transmit underflow interrupt enable
<i>ENET_DMA_INT_RI E</i>	receive interrupt enable
<i>ENET_DMA_INT_R BUIE</i>	receive buffer unavailable interrupt enable
<i>ENET_DMA_INT_R PSIE</i>	receive process stopped interrupt enable
<i>ENET_DMA_INT_R WTIE</i>	receive watchdog timeout interrupt enable
<i>ENET_DMA_INT_E TIE</i>	early transmit interrupt enable
<i>ENET_DMA_INT_F BEIE</i>	fatal bus error interrupt enable
<i>ENET_DMA_INT_E RIE</i>	early receive interrupt enable
<i>ENET_DMA_INT_AI E</i>	abnormal interrupt summary enable
<i>ENET_DMA_INT_NI E</i>	normal interrupt summary enable

Enum enet_int_flag_enum

Table 3-257. Enum enet_int_flag_enum

member name	Function description
<i>ENET_MAC_INT_F LAG_WUM</i>	WUM status flag
<i>ENET_MAC_INT_F LAG_MSC</i>	MSC status flag
<i>ENET_MAC_INT_F LAG_MSCR</i>	MSC receive status flag
<i>ENET_MAC_INT_F LAG_MSCT</i>	MSC transmit status flag
<i>ENET_MAC_INT_F LAG_TMST</i>	time stamp trigger status flag

<i>ENET_MSC_INT_F</i> <i>LAG_RFCE</i>	received frames CRC error flag
<i>ENET_MSC_INT_F</i> <i>LAG_RFAE</i>	received frames alignment error flag
<i>ENET_MSC_INT_F</i> <i>LAG_RGUF</i>	received good unicast frames flag
<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>	MSC status flag

<i>LAG_MSC</i>	
<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 3-258. `Enum enet_int_flag_clear_enum`

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

Enum enet_desc_reg_enum
Table 3-259. Enum enet_desc_reg_enum

member name	Function description
<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

Enum enet_msc_counter_enum
Table 3-260. Enum enet_msc_counter_enum

member name	Function description
<i>ENET_MSC_TX_SINGLE_COLLISION_CNT</i>	MSC transmitted good frames after a single collision counter
<i>ENET_MSC_TX_MORE_COLLISION_CNT</i>	MSC transmitted good frames after more than a single collision counter
<i>ENET_MSC_TX_GOOD_FRAME_CNT</i>	MSC transmitted good frames counter
<i>ENET_MSC_RX_CRC_ERROR_CNT</i>	MSC received frames with CRC error counter
<i>ENET_MSC_RX_ALIGNMENT_ERROR_CNT</i>	MSC received frames with alignment error counter
<i>ENET_MSC_RX_GOOD_UNICAST_FRAME_CNT</i>	MSC received good unicast frames counter

Enum enet_option_enum
Table 3-261. 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</i>	choose to configure the DMA arbitration related parameters

<i>N_OPTION</i>	
<i>STORE_OPTION</i>	choose to configure the store forward mode related parameters
<i>DMA_OPTION</i>	choose to configure the DMA related parameters
<i>VLAN_OPTION</i>	choose to configure vlan related parameters
<i>FLOWCTL_OPTION</i>	choose to configure flow control related parameters
<i>HASHH_OPTION</i>	choose to configure hash high
<i>HASHL_OPTION</i>	choose to configure hash low
<i>FILTER_OPTION</i>	choose to configure frame filter related parameters
<i>HALFDUPLEX_OPTION</i>	choose to configure halfduplex mode related parameters
<i>TIMER_OPTION</i>	choose to configure time counter related parameters
<i>INTERFRAMEGAP_OPTION</i>	choose to configure the inter frame gap related parameters

Enum `enet_mediemode_enum`

Table 3-262. `Enum enet_mediemode_enum`

member name	Function description
<i>ENET_AUTO_NEGOTIATION</i>	PHY auto negotiation
<i>ENET_100M_FULL DUPLEX</i>	100Mbit/s, full-duplex
<i>ENET_100M_HALF DUPLEX</i>	100Mbit/s, half-duplex
<i>ENET_10M_FULL DUPLEX</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 3-263. `Enum enet_chksumconf_enum`

member name	Function description
<i>ENET_NO_AUTOCHECKSUM</i>	disable IP frame checksum function
<i>ENET_AUTOCHECKSUM_DROP_FAIL_FRAMES</i>	enable IP frame checksum function
<i>ENET_AUTOCHECKSUM_ACCEPT_FAILED_FRAMES</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 3-264. Enum enet_frmrecept_enum

member name	Function description
<i>ENET_PROMISCIOUS_MODE</i>	promiscuous mode enabled
<i>ENET_RECEIVEALL_L</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

Enum enet_registers_type_enum
Table 3-265. Enum enet_registers_type_enum

member name	Function description
<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

Enum enet_dmadirection_enum
Table 3-266. 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 3-267. 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 3-268. Enum enet_regdirection_enum

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

member name	Function description
ENET_REG_WRITE	write register

Enum enet_macaddress_enum

Table 3-269. Enum enet_macaddress_enum

member name	Function description
<i>ENET_MAC_ADDR_ESS0</i>	set MAC address 0 filter
<i>ENET_MAC_ADDR_ESS1</i>	set MAC address 1 filter
<i>ENET_MAC_ADDR_ESS2</i>	set MAC address 2 filter
<i>ENET_MAC_ADDR_ESS3</i>	set MAC address 3 filter

Enum enet_descstate_enum

Table 3-270. 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

Enum enet_msc_preset_enum

Table 3-271. Enum enet_msc_preset_enum

member name	Function description
<i>ENET_MSC_PRES_ET_NONE</i>	do not preset MSC counter
<i>ENET_MSC_PRES_ET_HALF</i>	preset all MSC counters to almost-half(0x7FFF FFF0) value
<i>ENET_MSC_PRES_ET_FULL</i>	preset all MSC counters to almost-full(0xFFFF FFF0) value

enet_deinit

The description of enet_deinit is shown as below:

Table 3-272. 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 3-273. 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 3-261. Enum enet_option_enum only one parameter can be selected which is shown as below
<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
<i>HALFDUPLEX_OPTION_N</i>	choose to configure halfduplex mode related parameters
<i>TIMER_OPTION</i>	choose to configure time counter related parameters
<i>INTERFRAMEGAP_OPTION</i>	choose to configure the inter frame gap related parameters
Input parameter{in}	
para (the value according to the parameter option)	all the related values should be configured which are shown as below example: para = (value1 value2 value3...)
When value of parameter option is <i>FORWARD_OPTION</i>	
value1	<i>ENET_AUTO_PADCRC_DROP_ENABLE</i> / <i>ENET_AUTO_PADCRC_DROP_DISABLE</i>
value2	<i>ENET_FORWARD_ERRFRAMES_ENABLE</i> / <i>ENET_FORWARD_ERRFRAMES_DISABLE</i>
value3	<i>ENET_FORWARD_UNDERSZ_GOODFRAMES_ENABLE</i> / <i>ENET_FORWARD_UNDERSZ_GOODFRAMES_DISABLE</i>
value4	<i>ENET_FORWARD_UNDERSZ_GOODFRAMES_ENABLE</i> / <i>ENET_FORWARD_UNDERSZ_GOODFRAMES_DISABLE</i>
When value of parameter option is <i>DMABUS_OPTION</i>	
value1	<i>ENET_ADDRESS_ALIGN_ENABLE</i> / <i>ENET_ADDRESS_ALIGN_DISABLE</i>
value2	<i>ENET_FIXED_BURST_ENABLE</i> / <i>ENET_FIXED_BURST_DISABLE</i>
value3	<i>ENET_MIXED_BURST_ENABLE</i> / <i>ENET_MIXED_BURST_DISABLE</i>
When value of parameter option is <i>DMA_MAXBURST_OPTION</i>	
value1	<i>ENET_RXDP_1BEAT</i> / <i>ENET_RXDP_2BEAT</i> / <i>ENET_RXDP_4BEAT</i> / <i>ENET_RXDP_8BEAT</i> / <i>ENET_RXDP_16BEAT</i> / <i>ENET_RXDP_32BEAT</i> / <i>ENET_RXDP_4xPGBL_4BEAT</i> / <i>ENET_RXDP_4xPGBL_8BEAT</i> / <i>ENET_RXDP_4xPGBL_16BEAT</i> / <i>ENET_RXDP_4xPGBL_32BEAT</i> / <i>ENET_RXDP_4xPGBL_64BEAT</i> / <i>ENET_RXDP_4xPGBL_128BEAT</i>
value2	<i>ENET_PGBL_1BEAT</i> / <i>ENET_PGBL_2BEAT</i> / <i>ENET_PGBL_4BEAT</i> / <i>ENET_PGBL_8BEAT</i> / <i>ENET_PGBL_16BEAT</i> / <i>ENET_PGBL_32BEAT</i> / <i>ENET_PGBL_4xPGBL_4BEAT</i> / <i>ENET_PGBL_4xPGBL_8BEAT</i> / <i>ENET_PGBL_4xPGBL_16BEAT</i> / <i>ENET_PGBL_4xPGBL_32BEAT</i> / <i>ENET_PGBL_4xPGBL_64BEAT</i> / <i>ENET_PGBL_4xPGBL_128BEAT</i>
value3	<i>ENET_RXTX_DIFFERENT_PGBL</i> / <i>ENET_RXTX_SAME_PGBL</i>
When value of parameter option is <i>DMA_ARBITRATION_OPTION</i>	
value1	<i>ENET_ARBITRATION_RXPRIORTX</i>
value2	<i>ENET_ARBITRATION_RXTX_1_1</i> / <i>ENET_ARBITRATION_RXTX_2_1</i>

	<i>ENET_ARBITRATION_RXTX_3_1 / ENET_ARBITRATION_RXTX_4_1</i>
When value of parameter option is <i>STORE_OPTION</i>	
value1	<i>ENET_RX_MODE_STOREFORWARD / ENET_RX_MODE_CUTTHROUGH</i>
value2	<i>ENET_TX_MODE_STOREFORWARD / ENET_TX_MODE_CUTTHROUGH</i>
value3	<i>ENET_RX_THRESHOLD_64BYTES / ENET_RX_THRESHOLD_32BYTES / ENET_RX_THRESHOLD_96BYTES / ENET_RX_THRESHOLD_128BYTES</i>
value4	<i>ENET_TX_THRESHOLD_64BYTES / ENET_TX_THRESHOLD_128BYTES / ENET_TX_THRESHOLD_192BYTES / ENET_TX_THRESHOLD_256BYTES / ENET_TX_THRESHOLD_40BYTES / ENET_TX_THRESHOLD_32BYTES / ENET_TX_THRESHOLD_24BYTES / ENET_TX_THRESHOLD_16BYTES</i>
When value of parameter option is <i>DMA_OPTION</i>	
value1	<i>ENET_FLUSH_RXFRAME_ENABLE / ENET_FLUSH_RXFRAME_DISABLE</i>
value2	<i>ENET_SECONDFRAME_OPT_ENABLE / ENET_SECONDFRAME_OPT_DISABLE</i>
value3	<i>ENET_ENHANCED_DESCRIPTOR / ENET_NORMAL_DESCRIPTOR</i>
When value of parameter option is <i>VLAN_OPTION</i>	
value1	<i>ENET_VLANTAGCOMPARISON_12BIT / ENET_VLANTAGCOMPARISON_16BIT</i>
value2	<i>MAC_VLT_VLTI(regval)</i>
When value of parameter option is <i>FLOWCTL_OPTION</i>	
value1	<i>MAC_FCTL_PTU(regval)</i>
value2	<i>ENET_ZERO_QUANTA_PAUSE_ENABLE / ENET_ZERO_QUANTA_PAUSE_DISABLE</i>
value3	<i>ENET_PAUSETIME_MINUS4 / ENET_PAUSETIME_MINUS28 / ENET_PAUSETIME_MINUS144 / ENET_PAUSETIME_MINUS256</i>
value4	<i>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDetect / ENET_UNIQUE_PAUSEDetect</i>
value5	<i>ENET_RX_FLOWCONTROL_ENABLE / ENET_RX_FLOWCONTROL_DISABLE</i>
value6	<i>ENET_TX_FLOWCONTROL_ENABLE / ENET_TX_FLOWCONTROL_DISABLE</i>
When value of parameter option is <i>HASHH_OPTION</i>	
value1	<i>0x0~0xFFFF FFFFU</i>
When value of parameter option is <i>HASHL_OPTION</i>	
value1	<i>0x0~0xFFFF FFFFU</i>
When value of parameter option is <i>FILTER_OPTION</i>	
value1	<i>ENET_SRC_FILTER_NORMAL_ENABLE / ENET_SRC_FILTER_INVERSE_ENABLE / ENET_SRC_FILTER_DISABLE</i>
value2	<i>ENET_DEST_FILTER_INVERSE_ENABLE / ENET_DEST_FILTER_INVERSE_DISABLE</i>

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

Example:

```
/* config DMA option of the ENET */
enet_initpara_reset();
enet_initpara_config(DMA_OPTION, ENET_FLUSH_RXFRAME_ENABLE | ENET_SECONDFRAME_OPT_ENABLE | ENET_NORMAL_DESCRIPTOR);
```

enet_init

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

Table 3-274. Function *enet_init*

Function name	enet_init
----------------------	------------------

Function prototype	ErrStatus enet_init(enet_mediemode_enum mediemode, 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}	
mediemode	PHY mode and mac loopback configurations, refer to Table 3-262. Enum enet_mediemode enum , only one parameter can be selected
<i>ENET_AUTO_NEGOTIATION</i>	PHY auto negotiation
<i>ENET_100M_FULLDUPLEX</i>	100Mbit/s, full-duplex
<i>ENET_100M_HALFDUPLEX</i>	100Mbit/s, half-duplex
<i>ENET_10M_FULLDUPLEX</i>	10Mbit/s, full-duplex
<i>ENET_10M_HALFDUPLEX</i>	10Mbit/s, half-duplex
<i>ENET_LOOPBACKMODE</i>	MAC in loopback mode at the MII
Input parameter{in}	
checksum	IP frame checksum offload function, refer to Table 3-263. Enum enet_chksumconf enum , only one parameter can be selected
<i>ENET_NO_AUTOCHECKSUM</i>	disable IP frame checksum function
<i>ENET_AUTOCHECKSUM_DROP_FAILFRAMES</i>	enable IP frame checksum function
<i>ENET_AUTOCHECKSUM_ACCEPT_FAILFRAMES</i>	enable IP frame checksum function, and the received frame with only payload error but no other errors will not be dropped
Input parameter{in}	
recept	frame filter function, refer to Table 3-264. Enum enet_frmrecept enum , 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_DR
OP_FAILFRAMES, ENET_BROADCAST_FRAMES_PASS);
```

enet_software_reset

The description of enet_software_reset is shown as below:

Table 3-275. 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 3-276. 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 3-277. Function enet_descriptors_chain_init

Function name		enet_descriptors_chain_init
Function prototype		void enet_descriptors_chain_init(enet_dmadirection_enum direction);
Function descriptions		initialize the DMA Tx/Rx descriptors's parameters in chain mode
Precondition		-
The called functions		-
Input parameter{in}		
direction	the descriptors which users want to init, refer to Table 3-266. Enum enet_dmadirection_enum only one parameter can be selected which is shown as below	
<i>ENET_DMA_TX</i>	DMA Tx descriptors	
<i>ENET_DMA_RX</i>	DMA Rx descriptors	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

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

enet_descriptors_ring_init

The description of enet_descriptors_ring_init is shown as below:

Table 3-278. Function enet_descriptors_ring_init

Function name	enet_descriptors_ring_init
Function prototype	void enet_descriptors_ring_init(enet_dmadirection_enum direction);
Function descriptions	initialize the DMA Tx/Rx descriptors's parameters in ring mode
Precondition	-
The called functions	-
Input parameter{in}	
direction	the descriptors which users want to init, refer to Table 3-266. Enum enet_dmadirection_enum only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_descriptors_ring_init(ENET_DMA_TX);
```

enet_frame_receive

The description of enet_frame_receive is shown as below:

Table 3-279. 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 3-280. 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 3-281. Function enet_transmit_checksum_config

Function name	enet_transmit_checksum_config
Function prototype	ErrStatus 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 3-252. Structure enet_descriptors_struct
Input parameter{in}	
checksum	IP frame checksum configuration only one parameter can be selected which is shown as below
<i>ENET_CHECKSUM_DISABLE</i>	checksum insertion disabled
<i>ENET_CHECKSUM_IP_V4HEADER</i>	only IP header checksum calculation and insertion are enabled
<i>ENET_CHECKSUM_TCPUDPICMP_SEGMENT</i>	TCP/UDP/ICMP checksum insertion calculated but pseudo-header
<i>ENET_CHECKSUM_TCPUDPICMP_FULL</i>	TCP/UDP/ICMP checksum insertion fully calculated
Output parameter{out}	
Return value	
ErrStatus	ERROR or SUCCESS

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 3-282. 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 3-283. 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 3-284. 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 3-269. Enum enet_macaddress_enum only one parameter can be selected which is shown as below
<i>ENET_MAC_ADDRESS_S0</i>	set MAC address 0 filter
<i>ENET_MAC_ADDRESS_S1</i>	set MAC address 1 filter
<i>ENET_MAC_ADDRESS</i>	set MAC address 2 filter

S2	
ENET_MAC_ADDRESS	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 3-285. Function enet_mac_address_get

Function name	enet_mac_address_get
Function prototype	ErrStatus enet_mac_address_get(enet_macaddress_enum mac_addr, uint8_t paddr[], uint8_t bufsize);
Function descriptions	get MAC address
Precondition	-
The called functions	-
Input parameter{in}	
mac_addr	select which MAC address will be set, refer to Table 3-269. Enum enet_macaddress_enum only one parameter can be selected which is shown as below
ENET_MAC_ADDRESS_S0	set MAC address 0 filter
ENET_MAC_ADDRESS	set MAC address 1 filter

S1	
<i>ENET_MAC_ADDRESS</i> S2	set MAC address 2 filter
<i>ENET_MAC_ADDRESS</i> S3	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}
Input parameter{in}	
bufsize	refer to the size of the buffer which stores the MAC address
Return value	
-	-

Example:

```
/* get mac address */
enet_mac_address_get (ENET_MAC_ADDRESS0, netif->hwaddr, 0x100);
```

enet_flag_get

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

Table 3-286. Function enet_flag_get

Function name	enet_flag_get
Function prototype	FlagStatus enet_flag_get(enet_flag_enum enet_flag);
Function descriptions	get the ENET MAC/MSC/PTP/DMA status flag
Precondition	-
The called functions	-
Input parameter{in}	
enet_flag	ENET status flag, refer to Table 3-254. Enum enet_flag_enum only one parameter can be selected which is shown as below
<i>ENET_MAC_FLAG_MP</i> KR	magic packet received flag
<i>ENET_MAC_FLAG_W</i> UFR	wakeup frame received flag
<i>ENET_MAC_FLAG_FL</i> OWCONTROL	flow control status flag
<i>ENET_MAC_FLAG_W</i> UM	WUM status flag
<i>ENET_MAC_FLAG_MS</i> C	MSC status flag
<i>ENET_MAC_FLAG_MS</i> CR	MSC receive status flag

<i>ENET_MAC_FLAG_MS</i> <i>CT</i>	MSC transmit status flag
<i>ENET_MAC_FLAG_TM</i> <i>ST</i>	time stamp trigger status flag
<i>ENET_PTP_FLAG_TS</i> <i>SCO</i>	timestamp second counter overflow flag
<i>ENET_PTP_FLAG_TT</i> <i>M</i>	target time match flag
<i>ENET_MSC_FLAG_RF</i> <i>CE</i>	received frames CRC error flag
<i>ENET_MSC_FLAG_RF</i> <i>AE</i>	received frames alignment error flag
<i>ENET_MSC_FLAG_RG</i> <i>UF</i>	received good unicast frames flag
<i>ENET_MSC_FLAG_TG</i> <i>FSC</i>	transmitted good frames single collision flag
<i>ENET_MSC_FLAG_TG</i> <i>FMSC</i>	transmitted good frames more single collision flag
<i>ENET_MSC_FLAG_TG</i> <i>F</i>	transmitted good frames flag
<i>ENET_DMA_FLAG_TS</i>	transmit status flag
<i>ENET_DMA_FLAG_TP</i> <i>S</i>	transmit process stopped status flag
<i>ENET_DMA_FLAG_TB</i> <i>U</i>	transmit buffer unavailable status flag
<i>ENET_DMA_FLAG_TJ</i> <i>T</i>	transmit jabber timeout status flag
<i>ENET_DMA_FLAG_RO</i>	receive overflow status flag
<i>ENET_DMA_FLAG_TU</i>	transmit underflow status flag
<i>ENET_DMA_FLAG_RS</i>	receive status flag
<i>ENET_DMA_FLAG_RB</i> <i>U</i>	receive buffer unavailable status flag
<i>ENET_DMA_FLAG_RP</i> <i>S</i>	receive process stopped status flag
<i>ENET_DMA_FLAG_R</i> <i>WT</i>	receive watchdog timeout status flag
<i>ENET_DMA_FLAG_ET</i>	early transmit status flag
<i>ENET_DMA_FLAG_FB</i> <i>E</i>	fatal bus error status flag
<i>ENET_DMA_FLAG_ER</i>	early receive status flag
<i>ENET_DMA_FLAG_AI</i>	abnormal interrupt summary flag
<i>ENET_DMA_FLAG_NI</i>	normal interrupt summary flag
<i>ENET_DMA_FLAG_EB</i>	DMA error flag

<i>_DMA_ERROR</i>	
<i>ENET_DMA_FLAG_EB</i> <i>_TRANSFER_ERROR</i>	transfer error flag
<i>ENET_DMA_FLAG_EB</i> <i>_ACCESS_ERROR</i>	access error flag
<i>ENET_DMA_FLAG_MS</i> <i>C</i>	MSC status flag
<i>ENET_DMA_FLAG_W</i> <i>UM</i>	WUM status flag
<i>ENET_DMA_FLAG_TS</i> <i>T</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 3-287. 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 3-255. Enum enet_flag_clear_enum only one parameter can be selected which is shown as below
<i>ENET_DMA_FLAG_TS</i> <i>_CLR</i>	transmit status flag clear
<i>ENET_DMA_FLAG_TP</i> <i>S_CLR</i>	transmit process stopped status flag clear
<i>ENET_DMA_FLAG_TB</i> <i>U_CLR</i>	transmit buffer unavailable status flag clear
<i>ENET_DMA_FLAG_TJ</i> <i>T_CLR</i>	transmit jabber timeout status flag clear
<i>ENET_DMA_FLAG_RO</i> <i>_CLR</i>	receive overflow status flag clear

<i>ENET_DMA_FLAG_TU_CLR</i>	transmit underflow status flag clear
<i>ENET_DMA_FLAG_RS_CLR</i>	receive status flag clear
<i>ENET_DMA_FLAG_RB_U_CLR</i>	receive buffer unavailable status flag clear
<i>ENET_DMA_FLAG_RP_S_CLR</i>	receive process stopped status flag clear
<i>ENET_DMA_FLAG_R_WT_CLR</i>	receive watchdog timeout status flag clear
<i>ENET_DMA_FLAG_ET_CLR</i>	early transmit status flag clear
<i>ENET_DMA_FLAG_FB_E_CLR</i>	fatal bus error status flag clear
<i>ENET_DMA_FLAG_ER_CLR</i>	early receive status flag clear
<i>ENET_DMA_FLAG_AI_CLR</i>	abnormal interrupt summary flag clear
<i>ENET_DMA_FLAG_NI_CLR</i>	normal interrupt summary flag clear
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_interrupt_enable

The description of enet_interrupt_enable is shown as below:

Table 3-288. 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 3-256. Enum enet_int_enum only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_WUM</i>	WUM interrupt mask

IM	
<i>ENET_MAC_INT_TMS</i> <i>TIM</i>	timestamp trigger interrupt mask
<i>ENET_MSC_INT_RFC</i> <i>EIM</i>	received frame CRC error interrupt mask
<i>ENET_MSC_INT_RFA</i> <i>EIM</i>	received frames alignment error interrupt mask
<i>ENET_MSC_INT_RGU</i> <i>FIM</i>	received good unicast frames interrupt mask
<i>ENET_MSC_INT_TGF</i> <i>SCIM</i>	transmitted good frames single collision interrupt mask
<i>ENET_MSC_INT_TGF</i> <i>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:

```

/* 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 3-289. 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}	
<i>enet_int</i>	ENET interrupt, refer to Table 3-256. Enum enet_int_enum only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_WUM_IM</i>	WUM interrupt mask
<i>ENET_MAC_INT_TMS_TIM</i>	timestamp trigger interrupt mask
<i>ENET_MSC_INT_RFC_EIM</i>	received frame CRC error interrupt mask
<i>ENET_MSC_INT_RFA_EIM</i>	received frames alignment error interrupt mask
<i>ENET_MSC_INT_RGU_FIM</i>	received good unicast frames interrupt mask
<i>ENET_MSC_INT_TGF_SCIM</i>	transmitted good frames single collision interrupt mask
<i>ENET_MSC_INT_TGF_MSCIM</i>	transmitted good frames more single collision interrupt mask
<i>ENET_MSC_INT_TGFI_M</i>	transmitted good frames interrupt mask
<i>ENET_DMA_INT_TIE</i>	transmit interrupt enable
<i>ENET_DMA_INT_TPSI_E</i>	transmit process stopped interrupt enable
<i>ENET_DMA_INT_TBUI_E</i>	transmit buffer unavailable interrupt enable
<i>ENET_DMA_INT_TJTI_E</i>	transmit jabber timeout interrupt enable
<i>ENET_DMA_INT_ROIE</i>	receive overflow interrupt enable
<i>ENET_DMA_INT_TUIE</i>	transmit underflow interrupt enable
<i>ENET_DMA_INT_RIE</i>	receive interrupt enable
<i>ENET_DMA_INT_RBUI</i>	receive buffer unavailable interrupt enable

<i>E</i>	
<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 3-290. 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}	
<i>int_flag</i>	ENET interrupt flag, refer to Table 3-257. Enum enet_int_flag_enum only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_FLA</i> <i>G_WUM</i>	WUM status flag
<i>ENET_MAC_INT_FLA</i> <i>G_MSC</i>	MSC status flag
<i>ENET_MAC_INT_FLA</i> <i>G_MSCR</i>	MSC receive status flag
<i>ENET_MAC_INT_FLA</i> <i>G_MSCT</i>	MSC transmit status flag
<i>ENET_MAC_INT_FLA</i> <i>G_TMST</i>	time stamp trigger status flag

<i>ENET_MSC_INT_FLA</i> <i>G_RFCE</i>	received frames CRC error flag
<i>ENET_MSC_INT_FLA</i> <i>G_RFAE</i>	received frames alignment error flag
<i>ENET_MSC_INT_FLA</i> <i>G_RGUF</i>	received good unicast frames flag
<i>ENET_MSC_INT_FLA</i> <i>G_TGFSC</i>	transmitted good frames single collision flag
<i>ENET_MSC_INT_FLA</i> <i>G_TGFMSC</i>	transmitted good frames more single collision flag
<i>ENET_MSC_INT_FLA</i> <i>G_TGF</i>	transmitted good frames flag
<i>ENET_DMA_INT_FLA</i> <i>G_TS</i>	transmit status flag
<i>ENET_DMA_INT_FLA</i> <i>G_TPS</i>	transmit process stopped status flag
<i>ENET_DMA_INT_FLA</i> <i>G_TBU</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_FLA</i> <i>G_TJT</i>	transmit jabber timeout status flag
<i>ENET_DMA_INT_FLA</i> <i>G_RO</i>	receive overflow status flag
<i>ENET_DMA_INT_FLA</i> <i>G_TU</i>	transmit underflow status flag
<i>ENET_DMA_INT_FLA</i> <i>G_RS</i>	receive status flag
<i>ENET_DMA_INT_FLA</i> <i>G_RBU</i>	receive buffer unavailable status flag
<i>ENET_DMA_INT_FLA</i> <i>G_RPS</i>	receive process stopped status flag
<i>ENET_DMA_INT_FLA</i> <i>G_RWT</i>	receive watchdog timeout status flag
<i>ENET_DMA_INT_FLA</i> <i>G_ET</i>	early transmit status flag
<i>ENET_DMA_INT_FLA</i> <i>G_FBE</i>	fatal bus error status flag
<i>ENET_DMA_INT_FLA</i> <i>G_ER</i>	early receive status flag
<i>ENET_DMA_INT_FLA</i> <i>G_AI</i>	abnormal interrupt summary flag
<i>ENET_DMA_INT_FLA</i> <i>G_NI</i>	normal interrupt summary flag
<i>ENET_DMA_INT_FLA</i>	MSC status flag

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

Example:

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

enet_interrupt_flag_clear

The description of `enet_interrupt_flag_clear` is shown as below:

Table 3-291. Function `enet_interrupt_flag_clear`

Function name	<code>enet_interrupt_flag_clear</code>
Function prototype	<code>void enet_interrupt_flag_clear(enet_int_flag_clear_enum int_flag_clear);</code>
Function descriptions	clear ENET DMA interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
<i>int_flag_clear</i>	clear ENET interrupt flag, refer to Table 3-258. Enum enet int flag clear enum . only one parameter can be selected which is shown as below
<i>ENET_DMA_INT_FLA</i>	
<i>G_TS_CLR</i>	transmit status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_TPS_CLR</i>	transmit process stopped status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_TBU_CLR</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_TJT_CLR</i>	transmit jabber timeout status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_RO_CLR</i>	receive overflow status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_TU_CLR</i>	transmit underflow status flag
<i>ENET_DMA_INT_FLA</i>	
<i>G_RS_CLR</i>	receive status flag
<i>ENET_DMA_INT_FLA</i>	receive buffer unavailable status flag

<i>G_RBU_CLR</i>	
<i>ENET_DMA_INT_FLA</i> <i>G_RPS_CLR</i>	receive process stopped status flag
<i>ENET_DMA_INT_FLA</i> <i>G_RWT_CLR</i>	receive watchdog timeout status flag
<i>ENET_DMA_INT_FLA</i> <i>G_ET_CLR</i>	early transmit status flag
<i>ENET_DMA_INT_FLA</i> <i>G_FBE_CLR</i>	fatal bus error status flag
<i>ENET_DMA_INT_FLA</i> <i>G_ER_CLR</i>	early receive status flag
<i>ENET_DMA_INT_FLA</i> <i>G_AI_CLR</i>	abnormal interrupt summary flag
<i>ENET_DMA_INT_FLA</i> <i>G_NI_CLR</i>	normal interrupt summary flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear receive status flag bit */

enet_interrupt_flag_clear(ENET_DMA_INT_FLAG_RS);
```

enet_tx_enable

The description of `enet_tx_enable` is shown as below:

Table 3-292. Function `enet_tx_enable`

Function name	<code>enet_tx_enable</code>
Function prototype	<code>void enet_tx_enable(void);</code>
Function descriptions	ENET Tx function enable (include MAC and DMA module)
Precondition	-
The called functions	<code>enet_txfifo_flush()</code>
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 3-293. 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 3-294. 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 3-295. 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 3-296. 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 3-265. 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_FCTL
<i>ALL_MSC_REG</i>	get the registers within the offset scope range ENET_MSC_CTL to ENET_MSC_RGUFCTN
<i>ALL_PTP_REG</i>	get the registers within the offset scope range ENET_PTP_TSCTL to ENET_PTP_PPSCTL
<i>ALL_DMA_REG</i>	get the registers within the offset scope range ENET_DMA_BCTL to ENET_DMA_CRBADDR

Input parameter{in}	
num	the number of registers that the user want to get, (0 -- 54)
Output parameter{out}	
preg	the application buffer pointer for storing the register value
Return value	
-	-

Example:

```
/* get all mac registers value */

uint32_t register_buffer[5];

enet_registers_get(ALL_MAC_REG, 5, register_buffer);
```

enet_debug_status_get

The description of enet_debug_status_get is shown as below:

Table 3-297. Function enet_debug_status_get

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

NG	
<i>ENET_TXFIFO_NOT_EMPTY</i>	TxFIFO is not empty
<i>ENET_TXFIFO_FULL</i>	TxFIFO is full
Output parameter{out}	
-	-
Return value	
uint32_t	value of the status users want to get

Example:

```
/* get debug message of Rx FIFO state */
uint32_t debug_value;
debug_value = enet_debug_status_get (ENET_RXFIFO_STATE);
```

enet_address_filter_enable

The description of `enet_address_filter_enable` is shown as below:

Table 3-298. 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}	
<i>mac_addr</i>	select which MAC address will be enable refer to Table 3-269. Enum <code>enet_macaddress_enum</code> . only one parameter can be selected which is shown as below
<i>ENET_MAC_ADDRESS1</i>	enable MAC address 1 filter
<i>ENET_MAC_ADDRESS2</i>	enable MAC address 2 filter
<i>ENET_MAC_ADDRESS3</i>	enable MAC address 3 filter
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_address_filter_disable

The description of enet_address_filter_disable is shown as below:

Table 3-299. 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 3-269. Enum enet_macaddress_enum . only one parameter can be selected which is shown as below
ENET_MAC_ADDRESS1	enable MAC address 1 filter
ENET_MAC_ADDRESS2	enable MAC address 2 filter
ENET_MAC_ADDRESS3	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 3-300. Function enet_address_filter_config

Function name	enet_address_filter_config
Function prototype	void enet_address_filter_config(enet_macaddress_enum mac_addr, uint32_t addr_mask, uint32_t filter_type);
Function descriptions	configure the MAC address filter
Precondition	-
The called functions	-
Input parameter{in}	
mac_addr	select which MAC address will be enable, refer to Table 3-269. Enum enet_macaddress_enum . only one parameter can be selected which is shown as below

<i>ENET_MAC_ADDRESS1</i>	enable MAC address 1 filter
<i>ENET_MAC_ADDRESS2</i>	enable MAC address 2 filter
<i>ENET_MAC_ADDRESS3</i>	enable MAC address 3 filter
Input parameter{in}	
addr_mask	select which MAC address bytes will be mask one or more parameters can be selected which are shown as below
<i>ENET_ADDRESS_MASK_BYTE0</i>	mask ENET_MAC_ADDR1L[7:0] bits
<i>ENET_ADDRESS_MASK_BYTE1</i>	mask ENET_MAC_ADDR1L[15:8] bits
<i>ENET_ADDRESS_MASK_BYTE2</i>	mask ENET_MAC_ADDR1L[23:16] bits
<i>ENET_ADDRESS_MASK_BYTE3</i>	mask ENET_MAC_ADDR1L [31:24] bits
<i>ENET_ADDRESS_MASK_BYTE4</i>	mask ENET_MAC_ADDR1H [7:0] bits
<i>ENET_ADDRESS_MASK_BYTE5</i>	mask ENET_MAC_ADDR1H [15:8] bits
Input parameter{in}	
filter_type	select which MAC address filter type will be selected only one parameter can be selected which is shown as below
<i>ENET_ADDRESS_FILTER_ER_SA</i>	The MAC address is used to compared with the SA field of the received frame
<i>ENET_ADDRESS_FILTER_ER_DA</i>	The MAC address is used to compared with the DA field of the received frame
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_address_filter_config(ENET_MAC_ADDRESS1, ENET_ADDRESS_MASK_BYTE0 |
ENET_ADDRESS_MASK_BYTE1 | ENET_ADDRESS_MASK_BYTE2, ENET_ADDRESS_
_FILTER_DA);
```

enet_phy_config

The description of `enet_phy_config` is shown as below:

Table 3-301. 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 3-302. 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	refer to Table 3-267. Enum enet_phydirection enum , only one parameter can be selected which is shown as below
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 3-303. 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 3-304. 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 3-305. Function enet_forward_feature_enable

Function name	enet_forward_feature_enable
Function prototype	void enet_forward_feature_enable(uint32_t feature);
Function descriptions	enable ENET forward feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET forward mode one or more parameters can be selected which are shown as below
<i>ENET_AUTO_PADCR_C_DROP</i>	the function of the MAC strips the Pad/FCS field on received frames
<i>ENET_TYPEFRAME_C_RC_DROP</i>	the function that FCS field(last 4 bytes) of frame will be dropped before forwarding
<i>ENET_FORWARD_ER_RFRAMES</i>	the function that all frame received with error except runt error are forwarded to memory
<i>ENET_FORWARD_UNDERSZ_GOODFRAME_S</i>	the function that forwarding undersized good frames
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the function that forwarding undersized good frames */

enet_forward_feature_enable(ENET_FORWARD_UNDERSZ_GOODFRAMES);
```

enet_forward_feature_disable

The description of enet_forward_feature_disable is shown as below:

Table 3-306. Function `enet_forward_feature_disable`

Function name	enet_forward_feature_disable
Function prototype	<code>void enet_forward_feature_disable(uint32_t feature);</code>
Function descriptions	disable ENET forward feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET forward mode one or more parameters can be selected which are shown as below
<i>ENET_AUTO_PADCR_C_DROP</i>	the function of the MAC strips the Pad/FCS field on received frames
<i>ENET_TYPEFRAME_C_RC_DROP</i>	the function that FCS field(last 4 bytes) of frame will be dropped before forwarding
<i>ENET_FORWARD_ER_RFRAMES</i>	the function that all frame received with error except runt error are forwarded to memory
<i>ENET_FORWARD_UNDERSZ_GOODFRAMES_S</i>	the function that forwarding undersized good frames
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_filter_feature_enable

The description of `enet_filter_feature_enable` is shown as below:

 Table 3-307. Function `enet_filter_feature_enable`

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

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

Example:

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

enet_filter_feature_disable

The description of `enet_filter_feature_disable` is shown as below:

Table 3-308. Function `enet_filter_feature_disable`

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

<code>ENET_FILTER_MODE_EITHER</code>	HASH or perfect filter function
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_pauseframe_generate

The description of `enet_pauseframe_generate` is shown as below:

Table 3-309. 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 rval;
rval = enet_pauseframe_generate();
```

enet_pauseframe_detect_config

The description of `enet_pauseframe_detect_config` is shown as below:

Table 3-310. Function `enet_pauseframe_detect_config`

Function name	enet_pauseframe_detect_config
Function prototype	void enet_pauseframe_detect_config(uint32_t detect);
Function descriptions	configure the pause frame detect type
Precondition	-

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

Example:

```
/* config the pause frame detect type as ENET_UNIQUE_PAUSEDTECT */
enet_pauseframe_detect_config(ENET_UNIQUE_PAUSEDTECT);
```

enet_pauseframe_config

The description of `enet_pauseframe_config` is shown as below:

Table 3-311. Function `enet_pauseframe_config`

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

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_pauseframe_config(30, ENET_PAUSETIME_MINUS4);
```

enet_flowcontrol_threshold_config

The description of enet_flowcontrol_threshold_config is shown as below:

Table 3-312. Function enet_flowcontrol_threshold_config

Input parameter{in}	
deactive	the threshold of the deactive flow control, this value should always be less than active flow control value, only one parameter can be selected which is shown as below
<i>ENET_DEACTIVE_TH RESHOLD_256BYTES</i>	threshold level is 256 bytes
<i>ENET_DEACTIVE_TH RESHOLD_512BYTES</i>	threshold level is 512 bytes
<i>ENET_DEACTIVE_TH RESHOLD_768BYTES</i>	threshold level is 768 bytes
<i>ENET_DEACTIVE_TH RESHOLD_1024BYTE S</i>	threshold level is 1024 bytes
<i>ENET_DEACTIVE_TH RESHOLD_1280BYTE S</i>	threshold level is 1280 bytes
<i>ENET_DEACTIVE_TH RESHOLD_1536BYTE S</i>	threshold level is 1536 bytes
<i>ENET_DEACTIVE_TH RESHOLD_1792BYTE S</i>	threshold level is 1792 bytes
Input parameter{in}	
active	the threshold of the active flow control, only one parameter can be selected

	which is shown as below
<i>ENET_ACTIVE_THRE</i> <i>SHOLD_256BYTES</i>	threshold level is 256 bytes
<i>ENET_ACTIVE_THRE</i> <i>SHOLD_512BYTES</i>	threshold level is 512 bytes
<i>ENET_ACTIVE_THRE</i> <i>SHOLD_768BYTES</i>	threshold level is 768 bytes
<i>ENET_ACTIVE_THRE</i> <i>SHOLD_1024BYTES</i>	threshold level is 1024 bytes
<i>ENET_ACTIVE_THRE</i> <i>SHOLD_1280BYTES</i>	threshold level is 1280 bytes
<i>ENET_ACTIVE_THRE</i> <i>SHOLD_1536BYTES</i>	threshold level is 1536 bytes
<i>ENET_ACTIVE_THRE</i> <i>SHOLD_1792BYTES</i>	threshold level is 1792 bytes
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_flowcontrol_feature_enable

The description of `enet_flowcontrol_feature_enable` is shown as below:

Table 3-313. Function `enet_flowcontrol_feature_enable`

Function name	enet_flowcontrol_feature_enable
Function prototype	void enet_flowcontrol_feature_enable(uint32_t feature);
Function descriptions	enable ENET flow control feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET flow control mode one or more parameters can be selected which are shown as below
<i>ENET_ZERO_QUANT_A_PAUSE</i>	the automatic zero-quanta generation function
<i>ENET_TX_FLOWCONTROL</i>	the flow control operation in the MAC
<i>ENET_RX_FLOWCONTROL</i>	decoding function for the received pause frame and process it

<i>TROL</i>	
<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 3-314. 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_TROL</i>	the flow control operation in the MAC
<i>ENET_RX_FLOWCONTROL_TROL</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 3-315. Function enet_dmaprocess_state_get

Function name	enet_dmaprocess_state_get
Function prototype	uint32_t enet_dmaprocess_state_get(enet_dmadirection_enum direction);
Function descriptions	get the dma transmit/receive process state
Precondition	-
The called functions	-
Input parameter{in}	
direction	choose the direction of DMA process which users want to resume only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA transmit process
<i>ENET_DMA_RX</i>	DMA receive process
Output parameter{out}	
-	-
Return value	
uint32_t	state of dma process, the value range shows below: ENET_RX_STATE_STOPPED / ENET_RX_STATE_FETCHING / ENET_RX_STATE_WAITING / ENET_RX_STATE_SUSPENDED / ENET_RX_STATE_CLOSING / ENET_RX_STATE_QUEUEING / ENET_TX_STATE_STOPPED / ENET_TX_STATE_FETCHING / ENET_TX_STATE_WAITING / ENET_TX_STATE_READING / ENET_TX_STATE_SUSPENDED / ENET_TX_STATE_CLOSING

Example:

```

/* get the dma receive process state */

uint32_t 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 3-316. Function enet_dmaprocess_resume

Function name	enet_dmaprocess_resume
Function prototype	void enet_dmaprocess_resume(enet_dmadirection_enum direction);
Function descriptions	poll the DMA transmission/reception enable by writing any value to the

	ENET_DMA_TPEN/ENET_DMA_RPEN register, this will make the DMA to resume transmission/reception
Precondition	-
The called functions	-
Input parameter{in}	
direction	choose the direction of DMA process which users want to resume only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA transmit process
<i>ENET_DMA_RX</i>	DMA receive process
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_rxprocess_check_recovery

The description of `enet_rxprocess_check_recovery` is shown as below:

Table 3-317. Function `enet_rxprocess_check_recovery`

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

Example:

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

enet_txfifo_flush

The description of `enet_txfifo_flush` is shown as below:

Table 3-318. Function enet_txfifo_flush

Function name	enet_txfifo_flush
Function prototype	ErrStatus enet_txfifo_flush(void);
Function descriptions	flush the ENET transmit FIFO, and wait until the flush operation completes
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* flush the ENET transmit FIFO */

ErrStatus reval = ERROR;

reval = enet_txfifo_flush();
```

enet_current_desc_address_get

The description of enet_current_desc_address_get is shown as below:

Table 3-319. 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 3-259. Enum enet_desc_req_enum only one parameter can be selected which is shown as below
<i>ENET_RX_DESC_TAB</i> <i>LE</i>	the start address of the receive descriptor table
<i>ENET_RX_CURRENT_</i> <i>DESC</i>	the start descriptor address of the current receive descriptor read by the RxDMA controller
<i>ENET_RX_CURRENT_</i> <i>BUFFER</i>	the current receive buffer address being read by the RxDMA controller
<i>ENET_TX_DESC_TAB</i> <i>LE</i>	the start address of the transmit descriptor table
<i>ENET_TX_CURRENT_</i> <i>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	
<i>uint32_t</i>	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 3-320. Function *enet_desc_information_get*

Function name	enet_desc_information_get
Function prototype	<i>uint32_t enet_desc_information_get(enet_descriptors_struct *desc, enet_descstate_enum info_get);</i>
Function descriptions	get the Tx or Rx descriptor information
Precondition	-
The called functions	-
Input parameter{in}	
<i>desc</i>	the descriptor pointer which users want to get information, the structure members can refer to Table 3-252. Structure enet_descriptors_struct
Input parameter{in}	
<i>info_get</i>	the descriptor information type which is selected, refer to Table 3-270. Enum enet_descstate_enum only one parameter can be selected which is shown as below
<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
Output parameter{out}	

-	-
Return value	
uint32_t	descriptor information if value is 0xFFFFFFFFU, means the false input parameter

Example:

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

uint32_t raval;

raval = 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 3-321. Function `enet_missed_frame_counter_get`

Function name	enet_missed_frame_counter_get
Function prototype	void enet_missed_frame_counter_get(uint32_t *rxfifo_drop, uint32_t *rxdma_drop);
Function descriptions	get the number of missed frames during receiving
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
rxfifo_drop	pointer to the number of frames dropped by Rx FIFO
Output parameter{out}	
rxdma_drop	pointer to the number of frames missed by the Rx DMA 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 3-322. 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 3-252. 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_DB</i>	deferred	
<i>ENET_TDES0_UFE</i>	underflow error	
<i>ENET_TDES0_EXD</i>	excessive deferral	
<i>ENET_TDES0_VFRM</i>	VLAN frame	
<i>ENET_TDES0_ECO</i>	excessive collision	
<i>ENET_TDES0_LCO</i>	late collision	
<i>ENET_TDES0_NCA</i>	no carrier	
<i>ENET_TDES0_LCA</i>	loss of carrier	
<i>ENET_TDES0_IPPE</i>	IP payload error	
<i>ENET_TDES0_FRMF</i>	frame flushed	
<i>ENET_TDES0_JT</i>	jabber timeout	
<i>ENET_TDES0_ES</i>	error summary	
<i>ENET_TDES0_IPHE</i>	IP header error	
<i>ENET_TDES0_TTMSS</i>	transmit timestamp status	
<i>ENET_TDES0_TCHM</i>	the second address chained mode	
<i>ENET_TDES0_TERM</i>	transmit end of ring mode	
<i>ENET_TDES0_TTSEN</i>	transmit timestamp function enable	
<i>ENET_TDES0_DPAD</i>	disable adding pad	
<i>ENET_TDES0_DCRC</i>	disable CRC	
<i>ENET_TDES0_FSG</i>	first segment	
<i>ENET_TDES0_LSG</i>	last segment	
<i>ENET_TDES0_INTC</i>	interrupt on completion	
<i>ENET_TDES0_DAV</i>	DAV bit	
When type of parameter desc is RX		
<i>ENET_RDES0_PCERR</i>	payload checksum error	
<i>ENET_RDES0_CERR</i>	CRC error	
<i>ENET_RDES0_DBERR</i>	dribble bit error	
<i>ENET_RDES0_RERR</i>	receive error	
<i>ENET_RDES0_RWDT</i>	receive watchdog timeout	
<i>ENET_RDES0_FRMT</i>	frame type	
<i>ENET_RDES0_LCO</i>	late collision	

<i>ENET_RDES0_IPHER</i> <i>R</i>	IP frame header error
<i>ENET_RDES0_LDES</i>	last descriptor
<i>ENET_RDES0_FDES</i>	first descriptor
<i>ENET_RDES0_VTAG</i>	VLAN tag
<i>ENET_RDES0_OERR</i>	overflow error
<i>ENET_RDES0_LERR</i>	length error
<i>ENET_RDES0_SAFF</i>	SA filter fail
<i>ENET_RDES0_DERR</i>	descriptor error
<i>ENET_RDES0_ERRS</i>	error summary
<i>ENET_RDES0_DAFF</i>	destination address filter fail
<i>ENET_RDES0_DAV</i>	descriptor available
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the bit flag of ENET DMA descriptor */
FlagStatus reval;
reval = enet_desc_flag_get(p_txdesc, ENET_TDES0_TCHM);
```

enet_desc_flag_set

The description of `enet_desc_flag_set` is shown as below:

Table 3-323. 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 3-252. Structure <code>enet_descriptors_struct</code>
Input parameter{in}	
desc_flag (the value according to the parameter desc)	the bit flag of ENET DMA descriptor only one parameter can be selected which is shown as below
When type of parameter desc is TX	
<i>ENET_TDES0_VFRM</i>	VLAN frame
<i>ENET_TDES0_FRMF</i>	frame flushed
<i>ENET_TDES0_TCHM</i>	the second address chained mode

<i>ENET_TDES0_TERM</i>	transmit end of ring mode
<i>ENET_TDES0_TTSEN</i>	transmit timestamp function enable
<i>ENET_TDES0_DPAD</i>	disable adding pad
<i>ENET_TDES0_DCRC</i>	disable CRC
<i>ENET_TDES0_FSG</i>	first segment
<i>ENET_TDES0_LSG</i>	last segment
<i>ENET_TDES0_INTC</i>	interrupt on completion
<i>ENET_TDES0_DAV</i>	DAV bit
When type of parameter desc is RX	
<i>ENET_RDES0_DAV</i>	descriptor available
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_desc_flag_clear

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

Table 3-324. Function *enet_desc_flag_clear*

Function name	<i>enet_desc_flag_clear</i>
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 3-252. Structure <i>enet_descriptors_struct</i>
Input parameter{in}	
desc_flag (the value according to the parameter desc)	the bit flag of ENET DMA descriptor only one parameter can be selected which is shown as below
When type of parameter desc is TX	
<i>ENET_TDES0_VFRM</i>	VLAN frame
<i>ENET_TDES0_FRMF</i>	frame flushed
<i>ENET_TDES0_TCHM</i>	the second address chained mode
<i>ENET_TDES0_TERM</i>	transmit end of ring mode
<i>ENET_TDES0_TTSEN</i>	transmit timestamp function enable

<i>ENET_TDES0_DPAD</i>	disable adding pad
<i>ENET_TDES0_DCRC</i>	disable CRC
<i>ENET_TDES0_FSG</i>	first segment
<i>ENET_TDES0_LSG</i>	last segment
<i>ENET_TDES0_INTC</i>	interrupt on completion
<i>ENET_TDES0_DAV</i>	DAV bit
When type of parameter desc is RX	
<i>ENET_RDES0_DAV</i>	descriptor available
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_rx_desc_immediate_receive_complete_interrupt

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

Table 3-325. Function *enet_rx_desc_immediate_receive_complete_interrupt*

Function name	enet_rx_desc_immediate_receive_complete_interrupt
Function prototype	void enet_rx_desc_immediate_receive_complete_interrupt(enet_descriptors_struct *desc);
Function descriptions	when receiving completed, set RS bit in ENET_DMA_STAT register will immediately set
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to get flag, the structure members can refer to Table 3-252. Structure <i>enet_descriptors_struct</i>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_rx_desc_delay_receive_complete_interrupt

The description of enet_rx_desc_delay_receive_complete_interrupt is shown as below:

Table 3-326. Function enet_rx_desc_delay_receive_complete_interrupt

Function name	enet_rx_desc_delay_receive_complete_interrupt
Function prototype	void enet_rx_desc_delay_receive_complete_interrupt(enet_descriptors_struct *desc, uint32_t delay_time);
Function descriptions	when receiving completed, set RS bit in ENET_DMA_STAT register will be set after a configurable delay time
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to get flag, the structure members can refer to Table 3-252. Structure enet_descriptors_struct
Input parameter{in}	
delay_time	delay a time of 256*delay_time HCLK(0x00000000 - 0x000000FF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_rx_desc_delay_receive_complete_interrupt(p_rxdesc, 0x00000010);
```

enet_rxframe_drop

The description of enet_rxframe_drop is shown as below:

Table 3-327. 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 3-328. 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 3-329. Function `enet_dma_feature_disable`

Function name	enet_dma_feature_disable
Function prototype	void enet_dma_feature_disable(uint32_t feature);
Function descriptions	disable DMA feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of DMA mode one or more parameters can be selected which are shown as below

<i>ENET_NO_FLUSH_RXFRAME</i>	RxDMA does not flushes frames function
<i>ENET_SECONDFRAME_OPT</i>	TxDMA controller operate on second frame function
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_rx_desc_enhanced_status_get

The description of `enet_rx_desc_enhanced_status_get` is shown as below:

Table 3-330. Function `enet_rx_desc_enhanced_status_get`

Function name	enet_rx_desc_enhanced_status_get
Function prototype	uint32_t enet_rx_desc_enhanced_status_get(enet_descriptors_struct *desc, uint32_t desc_status);
Function descriptions	get the bit of extended status flag in ENET DMA descriptor
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to get the extended status flag, the structure members can refer to Table 3-252. Structure enet descriptors struct
Input parameter{in}	
desc_status	desc_status: the extended status want to get only one parameter can be selected which is shown as below
<i>ENET_RDES4_IPPLDT</i>	IP frame payload type
<i>ENET_RDES4_IPHERR</i>	IP frame header error
<i>ENET_RDES4_IPPLDERR</i>	IP frame payload error
<i>ENET_RDES4_IPCKSB</i>	IP frame checksum bypassed
<i>ENET_RDES4_IPF4</i>	IP frame in version 4
<i>ENET_RDES4_IPF6</i>	IP frame in version 6
<i>ENET_RDES4_PTPMT</i>	PTP message type
<i>ENET_RDES4_PTPOEF</i>	PTP on ethernet frame

ENET_RDES4_PTPVF	PTP version format
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* get the IP frame payload type in ENET DMA descriptor */
uint32_t status;
status = enet_rx_desc_enhanced_status_get(p_rxdesc, ENET_RDES4_IPPLDT);

enet_desc_select_enhanced_mode
```

The description of `enet_desc_select_enhanced_mode` is shown as below:

Table 3-331. Function `enet_desc_select_enhanced_mode`

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

Example:

```
/* configure descriptor to work in enhanced mode */
enet_desc_select_enhanced_mode();
```

`enet_ptp_enhanced_descriptors_chain_init`

The description of `enet_ptp_enhanced_descriptors_chain_init` is shown as below:

Table 3-332. Function `enet_ptp_enhanced_descriptors_chain_init`

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

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

Example:

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

enet_ptp_enhanced_descriptors_ring_init

The description of `enet_ptp_enhanced_descriptors_ring_init` is shown as below:

Table 3-333. Function `enet_ptp_enhanced_descriptors_ring_init`

Function name	enet_ptp_enhanced_descriptors_ring_init	
Function prototype	void enet_ptp_enhanced_descriptors_ring_init(enet_dmadirection_enum direction);	
Function descriptions	initialize the DMA Tx/Rx descriptors's parameters in enhanced ring mode with ptph 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	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

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

enet_ptpframe_receive_enhanced_mode

The description of enet_ptpframe_receive_enhanced_mode is shown as below:

Table 3-334. Function enet_ptpframe_receive_enhanced_mode

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

Example:

```

/* receive a packet data with timestamp values to application buffer in DMA enhanced mode
*/
uint32_t rx_buffer[500];
uint32_t time_stamp[2];
ErrStatus status;
status = enet_ptpframe_receive_enhanced_mode (rx_buffer, 500, time_stamp);

```

enet_ptpframe_transmit_enhanced_mode

The description of enet_ptpframe_transmit_enhanced_mode is shown as below:

Table 3-335. Function enet_ptpframe_transmit_enhanced_mode

Function name	enet_ptpframe_transmit_enhanced_mode
Function prototype	ErrStatus enet_ptpframe_transmit_enhanced_mode(uint8_t *buffer, uint32_t length, uint32_t timestamp[]);
Function descriptions	send data with timestamp values in application buffer as a transmit packet, when the DMA is in enhanced mode
Precondition	-
The called functions	-
Input parameter{in}	
buffer	pointer on the application buffer

	note -- if the input is NULL, user should copy data in application by himself
Input parameter{in}	
length	the length of frame data to be transmitted
Output parameter{out}	
timestamp	pointer to the table which stores the timestamp high and low note -- if the input is NULL, timestamp is ignored
Return value	
ErrStatus	SUCCESS or ERROR

Example:

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

uint32_t tx_buffer[500];
uint32_t time_stamp[2];
ErrStatus status;

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

enet_desc_select_normal_mode

The description of `enet_desc_select_normal_mode` is shown as below:

Table 3-336. Function `enet_desc_select_normal_mode`

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

Example:

```
/* configure descriptor to work in normal mode */

enet_desc_select_normal_mode( );
```

enet_ptp_normal_descriptors_chain_init

The description of `enet_ptp_normal_descriptors_chain_init` is shown as below:

Table 3-337. 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_dmadiraction_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
<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors
Input parameter{in}	
desc_ptptab	pointer to the first descriptor address of PTP Rx descriptor table, the structure members can refer to Table 3-252. 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 3-338. Function `enet_ptp_normal_descriptors_ring_init`

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

<i>ENET_DMA_RX</i>	DMA Rx descriptors
Input parameter{in}	
<i>desc_ptptab</i>	pointer to the first descriptor address of PTP Rx descriptor table, the structure members can refer to Table 3-252. 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 3-339. Function `enet_ptpframe_receive_normal_mode`

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

Example:

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

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

enet_ptpframe_transmit_normal_mode

The description of enet_ptpframe_transmit_normal_mode is shown as below:

Table 3-340. Function enet_ptpframe_transmit_normal_mode

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

Example:

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

uint32_t tx_buffer[500];

uint32_t time_stamp[2];

ErrStatus status;

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

enet_wum_filter_register_pointer_reset

The description of enet_wum_filter_register_pointer_reset is shown as below:

Table 3-341. 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 3-342. 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 3-343. Function `enet_wum_feature_enable`

Function name	enet_wum_feature_enable
Function prototype	void enet_wum_feature_enable(uint32_t feature);
Function descriptions	enable wakeup management features
Precondition	-

The called functions		-
Input parameter{in}		
feature		one or more parameters can be selected which are shown as below
<i>ENET_WUM_POWER_DOWN</i>		power down mode
<i>ENET_WUM_MAGIC_PACKET_FRAME</i>		enable a wakeup event due to magic packet reception
<i>ENET_WUM_WAKE_UP_FRAME</i>		enable a wakeup event due to wakeup frame reception
<i>ENET_WUM_GLOBAL_UNICAST</i>		any received unicast frame passed filter is considered to be a wakeup frame
Output parameter{out}		
-		-
Return value		
-		-

Example:

```
/* enable power down mode */
enet_wum_feature_enable(ENET_WUM_POWER_DOWN);
```

enet_wum_feature_disable

The description of `enet_wum_feature_disable` is shown as below:

Table 3-344. Function `enet_wum_feature_disable`

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

Example:

```

/* disable power down mode */

enet_wum_feature_disable(ENET_WUM_POWER_DOWN);

```

enet_msc_counters_reset

The description of `enet_msc_counters_reset` is shown as below:

Table 3-345. 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 3-346. Function `enet_msc_feature_enable`

Function name	enet_msc_feature_enable
Function prototype	void enet_msc_feature_enable(uint32_t feature);
Function descriptions	enable the MAC statistics counter features
Precondition	-
The called functions	-
Input parameter{in}	
feature	one or more parameters can be selected which are shown as below
<i>ENET_MSC_COUNTE R_STOP_ROLLOVER</i>	counter stop rollover
<i>ENET_MSC_RESET_O N_READ</i>	reset on read
<i>ENET_MSC_COUNTE RS_FREEZE</i>	MSC counter freeze
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* enable counter stop rollover function */
enet_msc_feature_enable(ENET_MSC_COUNTER_STOP_ROLLOVER);
```

enet_msc_feature_disable

The description of enet_msc_feature_disable is shown as below:

Table 3-347. 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_preset_config

The description of enet_msc_counters_preset_config is shown as below:

Table 3-348. Function enet_msc_counters_preset_config

Function name	enet_msc_counters_preset_config
Function prototype	void enet_msc_counters_preset_config(enet_msc_preset_enum mode);
Function descriptions	configure MAC statistics counters preset mode
Precondition	-
The called functions	-

Input parameter{in}	
mode	MSC counters preset mode, refer to Table 3-271. Enum enet_msc_preset_enum only one parameter can be selected which is shown as below
<i>ENET_MSC_PRESET_NONE</i>	do not preset MSC counter
<i>ENET_MSC_PRESET_HALF</i>	preset all MSC counters to almost-half(0x7FFF FFF0) value
<i>ENET_MSC_PRESET_FULL</i>	preset all MSC counters to almost-full(0xFFFF FFF0) value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* preset all MSC counters to almost-half */
enet_msc_counters_preset_config (ENET_MSC_PRESET_HALF);

enet_msc_counters_get
```

The description of `enet_msc_counters_get` is shown as below:

Table 3-349. 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, refer to Table 3-260. Enum enet_msc_counter_enum only one parameter can be selected which is shown as below
<i>ENET_MSC_TX_SCCN_T</i>	MSC transmitted good frames after a single collision counter
<i>ENET_MSC_TX_MSC_CNT</i>	MSC transmitted good frames after more than a single collision counter
<i>ENET_MSC_TX_TGFC_NT</i>	MSC transmitted good frames counter
<i>ENET_MSC_RX_RFCE_CNT</i>	MSC received frames with CRC error counter
<i>ENET_MSC_RX_RFAE_CNT</i>	MSC received frames with alignment error counter

<i>ENET_MSC_RX_RGU FCNT</i>	MSC received good unicast frames counter
Output parameter{out}	
-	-
Return value	
<i>uint32_t</i>	the MSC counter value

Example:

```
/* get MSC transmitted good frames after a single collision counter value*/
```

```
uin32_t reval;
```

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

enet_ptp_subsecond_2_nanosecond

The description of *enet_ptp_subsecond_2_nanosecond* is shown as below:

Table 3-350. Function enet_ptp_subsecond_2_nanosecond

Function name	enet_ptp_subsecond_2_nanosecond
Function prototype	<i>uint32_t enet_ptp_subsecond_2_nanosecond(uint32_t subsecond);</i>
Function descriptions	change subsecond to nanosecond
Precondition	-
The called functions	-
Input parameter{in}	
subsecond	subsecond value
Output parameter{out}	
-	-
Return value	
<i>uint32_t</i>	the nanosecond value

Example:

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

```
uin32_t reval;
```

```
reval = enet_ptp_subsecond_2_nanosecond (2);
```

enet_ptp_nanosecond_2_subsecond

The description of *enet_ptp_nanosecond_2_subsecond* is shown as below:

Table 3-351. Function enet_ptp_nanosecond_2_subsecond

Function name	enet_ptp_nanosecond_2_subsecond
Function prototype	<i>uint32_t enet_ptp_nanosecond_2_subsecond(uint32_t nanosecond);</i>
Function descriptions	change nanosecond to subsecond
Precondition	-

The called functions		-
Input parameter{in}		
nanosecond		Nanosecond value
Output parameter{out}		
-		-
Return value		
uint32_t		the subsecond value

Example:

```
/* change nanosecond to subsecond */

uint32_t reval;

reval = enet_ptp_nanosecond_2_subsecond (2);
```

enet_ptp_feature_enable

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

Table 3-352. Function enet_ptp_feature_enable

Function name	enet_ptp_feature_enable
Function prototype	void enet_ptp_feature_enable(uint32_t feature);
Function descriptions	enable the PTP features
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET PTP mode one or more parameters can be selected which are shown as below
<i>ENET_RXTX_TIMESTAMP</i> <i>AMP</i>	timestamp function for transmit and receive frames
<i>ENET_PTP_TIMESTAMP_INTERRUPT</i>	timestamp interrupt trigger
<i>ENET_ALL_RX_TIMESTAMP</i> <i>TAMP</i>	all received frames are taken snapshot
<i>ENET_NONTYPE_FRAME_SNAPSHOT</i>	take snapshot when received non type frame
<i>ENET_IPV6_FRAME_SNAPSHOT</i>	take snapshot for IPv6 frame
<i>ENET_IPV4_FRAME_SNAPSHOT</i>	take snapshot for IPv4 frame
<i>ENET_PTP_FRAME_USE_MACADDRESS_FILTER</i>	use MAC address1-3 to filter the PTP frame
Output parameter{out}	
-	-

Return value	
-	-

Example:

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

enet_ptp_feature_disable

The description of enet_ptp_feature_disable is shown as below:

Table 3-353. 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
ENET_RXTX_TIMESTAMP	timestamp function for transmit and receive frames
AMP	
ENET_PTP_TIMESTAMP_MP_INT	timestamp interrupt trigger
ENET_ALL_RX_TIMESTAMP	all received frames are taken snapshot
ENET_NONTYPE_FRAME_SNAPSHOT	take snapshot when received non type frame
ENET_IPV6_FRAME_SNAPSHOT	take snapshot for IPv6 frame
ENET_IPV4_FRAME_SNAPSHOT	take snapshot for IPv4 frame
ENET_PTP_FRAME_USE_MACADDRESS_FILTER	use MAC address1-3 to filter the PTP frame
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable PTP function for all received frames */
enet_ptp_feature_disable(ENET_ALL_RX_TIMESTAMP);
```

enet_ptp_timestamp_function_config

The description of enet_ptp_timestamp_function_config is shown as below:

Table 3-354. 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_CKNT_ORDINARY</i>	type of ordinary clock node type for timestamp
<i>ENET_CKNT_BOUNDARY</i>	type of boundary clock node type for timestamp
<i>ENET_CKNT_END_TO_END</i>	type of end-to-end transparent clock node type for timestamp
<i>ENET_CKNT_PEER_TO_PEER</i>	type of peer-to-peer transparent clock node type for timestamp
<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_FINE_MODE_E</i>	the system timestamp uses the fine method for updating
<i>ENET_PTP_COARSE_MODE</i>	the system timestamp uses the coarse method for updating
<i>ENET_SUBSECOND_DIGITAL_ROLLOVER</i>	digital rollover mode
<i>ENET_SUBSECOND_BINARY_ROLLOVER</i>	digital rollover mode
<i>ENET_SNOOPING_PT_P_VERSION_2</i>	version 2
<i>ENET_SNOOPING_PT_P_VERSION_1</i>	version 1
<i>ENET_EVENT_TYPE_MESSAGES_SNAPSHOT</i>	only event type messages are taken snapshot
<i>ENET_ALL_TYPE_MESSAGES_SNAPSHOT</i>	all type messages are taken snapshot except announce, management and signaling message

<i>ENET_MASTER_NOD E_MESSAGE_SNAPS HOT</i>	snapshot is only take for master node message
<i>ENET_SLAVE_NODE_ MESSAGE_SNAPSHO T</i>	snapshot is only taken for slave node message
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* config addend register update function */
ErrStatus reval = ERROR;
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 3-355. 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 3-356. Function enet_ptp_timestamp_addend_config

Function name	enet_ptp_timestamp_addend_config
Function prototype	void enet_ptp_timestamp_addend_config(uint32_t add);
Function descriptions	adjusting the clock frequency only in fine update mode
Precondition	-
The called functions	-
Input parameter{in}	
add	the value will be added to the accumulator register to achieve time synchronization (0 – 0xFFFF FFFF)
Output parameter{out}	
-	-
Return value	
-	-

/* added 0x1FFF to the accumulator register */

```
enet_ptp_timestamp_addend_config(0x1FFF);
```

enet_ptp_timestamp_update_config

The description of enet_ptp_timestamp_update_config is shown as below:

Table 3-357. Function enet_ptp_timestamp_update_config

Function name	enet_ptp_timestamp_update_config
Function prototype	void enet_ptp_timestamp_update_config(uint32_t sign, uint32_t second, uint32_t subsecond);
Function descriptions	initialize or add/subtract to second of the system time
Precondition	-
The called functions	-
Input parameter{in}	
sign	timestamp update positive or negative sign, only one parameter can be selected which is shown as below
<i>ENET_PTP_ADD_TO_TIME</i>	update value is added to system time
<i>ENET_PTP_SUBSTRACT_FROM_TIME</i>	timestamp update value is subtracted from system time
Input parameter{in}	
second	initializing or adding/subtracting to second of the system time (0 – 0xFFFF FFFF)
Input parameter{in}	
subsecond	the current subsecond of the system time with 0.46 ns accuracy (0 – 0x7FFF FFFF)
Output parameter{out}	
-	-
Return value	

-	-	-
---	---	---

Example:

```
/* initialize system time with timestamp update value */

enet_ptp_timestamp_update_config(ENET_PTP_ADD_TO_TIME, 0, 0);
```

enet_ptp_expected_time_config

The description of enet_ptp_expected_time_config is shown as below:

Table 3-358. 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 3-359. 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 3-253. 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_pps_output_frequency_config

The description of enet_ptp_pps_output_frequency_config is shown as below:

Table 3-360. enet_ptp_pps_output_frequency_config

Function name	enet_ptp_pps_output_frequency_config
Function prototype	void enet_ptp_pps_output_frequency_config(uint32_t freq);
Function descriptions	configure the PPS output frequency
Precondition	-
The called functions	-
Input parameter{in}	
freq	
<i>ENET_PPSOFC_1HZ</i>	PPS output 1Hz frequency
<i>ENET_PPSOFC_2HZ</i>	PPS output 2Hz frequency
<i>ENET_PPSOFC_4HZ</i>	PPS output 4Hz frequency
<i>ENET_PPSOFC_8HZ</i>	PPS output 8Hz frequency
<i>ENET_PPSOFC_16HZ</i>	PPS output 16Hz frequency
<i>ENET_PPSOFC_32HZ</i>	PPS output 32Hz frequency
<i>ENET_PPSOFC_64HZ</i>	PPS output 64Hz frequency
<i>ENET_PPSOFC_128HZ</i> <i>Z</i>	PPS output 128Hz frequency
<i>ENET_PPSOFC_256HZ</i> <i>Z</i>	PPS output 256Hz frequency
<i>ENET_PPSOFC_512HZ</i> <i>Z</i>	PPS output 512Hz frequency
<i>ENET_PPSOFC_1024HZ</i>	PPS output 1024Hz frequency
<i>ENET_PPSOFC_2048HZ</i>	PPS output 2048Hz frequency
<i>ENET_PPSOFC_4096HZ</i>	PPS output 4096Hz frequency
<i>ENET_PPSOFC_8192HZ</i>	PPS output 8192Hz frequency

<i>ENET_PPSOFC_16384HZ</i>	PPS output 16384Hz frequency
<i>ENET_PPSOFC_32768HZ</i>	PPS output 32768Hz frequency
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PPS output frequency as 1Hz */
enet_ptp_pps_output_frequency_config(ENET_PPSOFC_1HZ);
```

enet_ptp_start

The enet_ptp_start is shown as below:

Table 3-361. 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	-
Input parameter{in}	
updatemethod	method for updating
<i>ENET_PTP_FINEMOD_E</i>	fine correction method
<i>ENET_PTP_COARSE_MODE</i>	coarse correction method
Input parameter{in}	
init_sec	second value for initializing system time
Input parameter{in}	
init_subsec	subsecond value for initializing system time
Input parameter{in}	
carry_cfg	the value to be added to the accumulator register (in fine method is used)
Input parameter{in}	
accuracy_cfg	the value to be added to the subsecond value of system time
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* gconfigure and start PTP timestamp counter*/

enet_ptp_start(ENET_PTP_FINEMODE, 10, 10, 10, 10);

```

enet_ptp_finecorrection_adjfreq

The enet_ptp_finecorrection_adjfreq is shown as below:

Table 3-362. 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	-
Input parameter{in}	
carry_cfg	the value to be added to the accumulator register
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* adjust frequency in fine method by configure addend register */

enet_ptp_finecorrection_adjfreq(10);

```

enet_ptp_coarsecorrection_systime_update

The enet_ptp_coarsecorrection_systime_update is shown as below:

Table 3-363. 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	-
Input parameter{in}	
systime_struct	the descriptor pointer which users want to configure, the structure members can refer to Table 3-253. Structure enet_ptp_systime_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* update system time in coarse method */

enet_ptp_systime_struct systime_struct;

enet_ptp_coarsecorrection_systime_update (&systime_struct);

```

enet_ptp_finecorrection_settime

The enet_ptp_finecorrection_settime is shown as below:

Table 3-364. 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	-
Input parameter{in}	
systime_struct	the descriptor pointer which users want to configure, the structure members can refer to Table 3-253. Structure enet_ptp_systime_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* set system time in fine method */

enet_ptp_systime_struct systime_struct;

enet_ptp_finecorrection_settime (&systime_struct);

```

enet_ptp_flag_get

The enet_ptp_flag_get is shown as below:

Table 3-365. 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_	timestamp update

UPDATE	
ENET_PTP_SYSTIME_INIT	timestamp initialize
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

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

FlagStatus status = enet_ptp_flag_get(ENET_PTP_ADDEND_UPDATE);
```

enet_initpara_reset

The description of enet_initpara_reset is shown as below:

Table 3-366. 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 3-367. EXMC Registers

Registers	Descriptions
EXMC_SNCTL	SRAM/NOR Flash control registers
EXMC_SNTCFG	SRAM/NOR Flash timing configuration registers
EXMC_SNWTCFG	SRAM/NOR Flash write timing configuration registers
EXMC_NPCTL	NAND flash/PC card control registers
EXMC_NPINTEN	NAND flash/PC card interrupt enable registers
EXMC_NPCTCFG	NAND flash/PC card common space timing configuration registers
EXMC_NPATCFG	NAND flash/PC card attribute space timing configuration registers
EXMC_PIOTCFG3	PC card I/O space timing configuration register
EXMC_NECC	NAND flash ECC registers

3.12.2. Descriptions of Peripheral functions

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

Table 3-368. EXMC firmware function

Function name	Function description
exmc_norsram_deinit	deinitialize EXMC NOR/SRAM region x
exmc_norsram_struct_para_init	initialize exmc_norsram_parameter_struct with the default values
exmc_norsram_init	initialize EXMC NOR/SRAM region x
exmc_norsram_enable	enable EXMC NOR/PSRAM region x
exmc_norsram_disable	disable EXMC NOR/PSRAM region x
exmc_norsram_page_size_config	configure CRAM page size
exmc_nand_deinit	deinitialize EXMC NAND bank x
exmc_nand_struct_para_init	initialize exmc_nand_parameter_struct with the default values
exmc_nand_init	initialize EXMC NAND bank x
exmc_nand_enable	enable EXMC NAND bank x
exmc_nand_disable	disable EXMC NAND bank x
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 EXMC PC card bank
exmc_pccard_disable	disable EXMC PC card bank
exmc_interrupt_enable	enable EXMC interrupt
exmc_interrupt_disable	disable EXMC interrupt
exmc_flag_get	get EXMC flag status
exmc_flag_clear	clear EXMC flag status

Function name	Function description
exmc_interrupt_flag_get	get EXMC interrupt flag
exmc_interrupt_flag_clear	clear EXMC interrupt flag

Structure exmc_norsram_timing_parameter_struct

Table 3-369. Structure exmc_norsram_timing_parameter_struct

Member name	Function description
asyn_access_mode	asynchronous access mode
syn_data_latency	configure the data latency, synchronous access mode valid
syn_clk_division	configure the clock divide ratio, synchronous access mode valid
bus_latency	configure the bus latency
asyn_data_setuptime	configure the data setup time, asynchronous access mode valid
asyn_address_holdtime	configure the address hold time, asynchronous access mode valid
asyn_address_setuptime	configure the data setup time, asynchronous access mode valid

Structure exmc_norsram_parameter_struct

Table 3-370. Structure exmc_norsram_parameter_struct

Member name	Function description
norsram_region	select the region of EXMC NOR/SRAM
write_mode	the write mode, synchronous mode or asynchronous mode
extended_mode	enable or disable the extended mode
asyn_wait	enable or disable the asynchronous wait function
nwait_signal	enable or disable the NWAIT signal while in synchronous burst mode
memory_write	enable or disable the write operation
nwait_config	NWAIT signal configuration, only work in synchronous mode
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 3-369. Structure exmc_norsram_timing_parameter_struct
write_timing	timing parameters for write when the extended mode is used, the structure members can refer to Table 3-369. Structure exmc_norsram_timing_parameter_struct

Structure exmc_nand_pccard_timing_parameter_struct

Table 3-371. Structure exmc_nand_pccard_timing_parameter_struct

Member name	Function description
databus_hiztime	configure the dadtabus 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 3-372. 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 3-371. Structure exmc_nand_pccard_timing_parameter_struct
attribute_space_timing	the timing parameters for NAND flash attribute space, the structure members can refer to Table 3-371. Structure exmc_nand_pccard_timing_parameter_struct

Structure exmc_pccard_parameter_struct

Table 3-373. Structure exmc_pccard_parameter_struct

Member name	Function description
atr_latency	configure the latency of ALE low to RB low
ctr_latency	configure the latency of CLE low to RB low
wait_feature	enable or disable the wait feature
common_space_timing	the timing parameters for PC card common space, the structure members can refer to Table 3-371. Structure exmc_nand_pccard_timing_parameter_struct
attribute_space_timing	the timing parameters for PC card attribute space, the structure members can refer to Table 3-371. Structure exmc_nand_pccard_timing_parameter_struct
io_space_timing	the timing parameters for PC card IO space, the structure members can refer to Table 3-371. Structure exmc_nand_pccard_timing_parameter_struct

exmc_norsram_deinit

The description of exmc_norsram_deinit is shown as below:

Table 3-374. 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	EXMC NOR/SRAM region
EXMC_BANK0_NORS RAM_REGIONx	x=0,1,2,3
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize the EXMC NOR/SRAM region1 of bank0 */
exmc_norsram_deinit(EXMC_BANK0_NORSRAM_REGION1);
```

exmc_norsram_struct_para_init

The description of exmc_norsram_struct_para_init is shown as below:

Table 3-375. Function exmc_norsram_struct_para_init

Function name	exmc_norsram_struct_para_init
Function prototype	void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
Function descriptions	initialize the struct exmc_norsram_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
exmc_norsram_init_struct	Structure for initialization, the structure members can refer to Table 3-370. Structure exmc_norsram_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize the struct nor_init_struct */

exmc_norsram_parameter_struct nor_init_struct;
exmc_norsram_struct_para_init (&nor_init_struct);

```

exmc_norsram_init

The description of exmc_norsram_init is shown as below:

Table 3-376. 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 3-370. Structure exmc_norsram_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize EXMC NOR/SRAM bank */

exmc_norsram_parameter_struct lcd_init_struct;
exmc_norsram_timing_parameter_struct lcd_timing_init_struct;

/* configure timing parameter */

lcd_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;
lcd_timing_init_struct.syn_data_latency = EXMC_DATALAT_2_CLK;
lcd_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;
lcd_timing_init_struct.bus_latency = 1;
lcd_timing_init_struct.asyn_data_setuptime = 5;
lcd_timing_init_struct.asyn_address_holdtime = 2;
lcd_timing_init_struct.asyn_address_setuptime = 2;

/* configure EXMC bus parameters */

lcd_init_struct.norsram_region = EXMC_BANK0_NORSRAM_REGION1;

```

```

lcd_init_struct.write_mode = EXMC_ASYN_WRITE;

lcd_init_struct.extended_mode = DISABLE;

lcd_init_struct.asyn_wait = DISABLE;

lcd_init_struct.nwait_signal = DISABLE;

lcd_init_struct.memory_write = ENABLE;

lcd_init_struct.nwait_config = EXMC_NWAIT_CONFIG_BEFORE;

lcd_init_struct.wrap_burst_mode = DISABLE;

lcd_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;

lcd_init_struct.burst_mode = DISABLE;

lcd_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;

lcd_init_struct.memory_type = EXMC_MEMORY_TYPE_SRAM;

lcd_init_struct.address_data_mux = DISABLE;

lcd_init_struct.read_write_timing = &lcd_timing_init_struct;

lcd_init_struct.write_timing = &lcd_timing_init_struct;

exmc_norsram_init(&lcd_init_struct);
  
```

exmc_norsram_enable

The description of exmc_norsram_enable is shown as below:

Table 3-377. Function exmc_norsram_enable

Function name	exmc_norsram_enable
Function prototype	void exmc_norsram_enable(uint32_t exmc_norsram_region);
Function descriptions	enable EXMC NOR/PSRAM region
Precondition	-
The called functions	-
Input parameter{in}	
exmc_norsram_region	EXMC NOR/SRAM region
EXMC_BANK0_NORS RAM_REGIONx	x=0,1,2,3
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the EXMC NOR/SRAM region1 of bank0 */

exmc_norsram_enable(EXMC_BANK0_NORSRAM_REGION1);
```

exmc_norsram_disable

The description of exmc_norsram_disable is shown as below:

Table 3-378. Function exmc_norsram_disable

Function name	exmc_norsram_disable
Function prototype	void exmc_norsram_disable(uint32_t exmc_norsram_region);
Function descriptions	disable EXMC NOR/PSRAM region
Precondition	-
The called functions	-
Input parameter{in}	
exmc_norsram_region	EXMC NOR/SRAM region
EXMC_BANK0_NORSRAM_REGIONx	x=0,1,2,3
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the EXMC NOR/SRAM region1 of bank0 */

exmc_norsram_disable(EXMC_BANK0_NORSRAM_REGION1);
```

exmc_norsram_page_size_config

The description of exmc_norsram_page_size_config is shown as below:

Table 3-379. Function exmc_norsram_page_size_config

Function name	exmc_norsram_page_size_config
Function prototype	void exmc_norsram_page_size_config(uint32_t page_size);
Function descriptions	configure CRAM page size
Precondition	-
The called functions	-
Input parameter{in}	
page_size	CRAM page size
EXMC_CRAM_AUTO_SPLIT	the clock is generated only during synchronous access
EXMC_CRAM_PAGE_SIZE_128_BYTES	page size is 128 bytes
EXMC_CRAM_PAGE_	page size is 256 bytes

<i>SIZE_256_BYTES</i>	
<i>EXMC_CRAM_PAGE_SIZE_512_BYTES</i>	page size is 512 bytes
<i>EXMC_CRAM_PAGE_SIZE_1024_BYTES</i>	page size is 1024 bytes
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CRAM page size */

exmc_norsram_page_size_config (EXMC_CRAM_PAGE_SIZE_128_BYTES);
```

exmc_nand_deinit

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

Table 3-380. Function exmc_nand_deinit

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

Example:

```
/* deinitialize EXMC NOR/SRAM bank1 */

exmc_norsram_deinit(EXMC_BANK1_NAND);
```

exmc_nand_struct_para_init

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

Table 3-381. Function exmc_nand_struct_para_init

Function name	exmc_nand_struct_para_init
Function prototype	void exmc_nand_struct_para_init(exmc_nand_parameter_struct* exmc_nand_init_struct);

Function descriptions	initialize the struct exmc_nand_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_init_struct t	Structure for initialization, the structure members can refer to Table 3-372. Structure exmc_nand_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the struct nand_init_struct */

exmc_nand_parameter_struct nand_init_struct;
exmc_nand_struct_para_init (&nand_init_struct);
```

exmc_nand_init

The description of exmc_nand_init is shown as below:

Table 3-382. 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 3-372. Structure exmc_nand_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
exmc_nand_parameter_struct nand_init_struct;
exmc_nand_pccard_timing_parameter_struct nand_timing_init_struct;
/* EXMC configuration */
nand_timing_init_struct.setuptime = 5;
nand_timing_init_struct.waittime = 4;
```

```

nand_timing_init_struct.holdtime = 2;

nand_timing_init_struct.databus_hiztime = 2;

nand_init_struct.nand_bank = EXMC_BANK1_NAND;

nand_init_struct.ecc_size = EXMC_ECC_SIZE_2048BYTES;

nand_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;

nand_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;

nand_init_struct.ecc_logic = ENABLE;

nand_init_struct.databus_width = EXMC_NAND_DATABUS_WIDTH_8B;

nand_init_struct.wait_feature = ENABLE;

nand_init_struct.common_space_timing = &nand_timing_init_struct;

nand_init_struct.attribute_space_timing = &nand_timing_init_struct;

exmc_nand_init(&nand_init_struct);
  
```

exmc_nand_enable

The description of exmc_nand_enable is shown as below:

Table 3-383. Function exmc_nand_enable

Function name	exmc_nand_enable
Function prototype	void exmc_nand_enable(uint32_t exmc_nand_bank);
Function descriptions	enable EXMC NAND bank
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_bank	the bank of NAND
EXMC_BANKx_NAND	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable EXMC NAND bank1 */

exmc_nand_enable(EXMC_BANK1_NAND);
  
```

exmc_nand_disable

The description of exmc_nand_disable is shown as below:

Table 3-384. Function exmc_nand_disable

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

Example:

```
/* disable EXMC NAND bank1 */

exmc_nand_disable(EXMC_BANK1_NAND);
```

exmc_nand_ecc_config

The description of exmc_nand_ecc_config is shown as below:

Table 3-385. Function exmc_nand_ecc_config

Function name	exmc_nand_ecc_config
Function prototype	void exmc_nand_ecc_config(uint32_t exmc_nand_bank, ControlStatus newvalue);
Function descriptions	enable or disable the EXMC NAND ECC function
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_bank	specifie the NAND bank
EXMC_BANKx_NAND	x=1,2
Input parameter{in}	
newvalue	ENABLE or DISABLE
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the EXMC NAND ECC function */

exmc_nand_ecc_config(EXMC_BANK1_NAND, ENABLE);
```

exmc_ecc_get

The description of exmc_ecc_get is shown as below:

Table 3-386. Function exmc_ecc_get

Function name	exmc_ecc_get
Function prototype	uint32_t exmc_ecc_get(uint32_t exmc_nand_bank);
Function descriptions	get the EXMC ECC value
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_bank	specifie the NAND bank
EXMC_BANKx_NAND	x=1,2
Output parameter{out}	
-	-
Return value	
uint32_t	the error correction code(ECC) value

Example:

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

uint32_t ecc_value;

ecc_value = exmc_ecc_get(EXMC_BANK1_NAND);
```

exmc_pccard_deinit

The description of exmc_pccard_deinit is shown as below:

Table 3-387. 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 3-388. Function exmc_pccard_struct_para_init

Function name	exmc_pccard_struct_para_init
Function prototype	void exmc_pccard_struct_para_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);
Function descriptions	initialize the struct exmc_pccard_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
exmc_pccard_init_str uct	Structure for initialization, the structure members can refer to Table 3-373. Structure exmc_pccard_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the struct pccard_init_struct */

exmc_pccard_parameter_struct pccard_init_struct;
exmc_pccard_struct_para_init (&pccard_init_struct);
```

exmc_pccard_init

The description of exmc_pccard_init is shown as below:

Table 3-389. 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_str uct	Structure for initialization, the structure members can refer to Table 3-373. Structure exmc_pccard_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

exmc_pccard_parameter_struct pccard_init_struct;
exmc_nand_pccard_timing_parameter_struct pccard_timing_init_struct;
/* EXMC configuration */

pccard_timing_init_struct.setuptime = 5;
pccard_timing_init_struct.waittime = 4;
pccard_timing_init_struct.holdtime = 2;
pccard_timing_init_struct.databus_hiztime = 2;
pccard_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;
pccard_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;
pccard_init_struct.wait_feature = ENABLE;
pccard_init_struct.common_space_timing = & pccard_timing_init_struct;
pccard_init_struct.attribute_space_timing = & pccard_timing_init_struct;
pccard_init_struct.io_space_timing = & pccard_timing_init_struct;
exmc_pccard_init(&pccard_init_struct);

```

exmc_pccard_enable

The description of exmc_pccard_enable is shown as below:

Table 3-390. Function exmc_pccard_enable

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

Example:

```

/* enable EXMC PC card bank */

exmc_pccard_enable();

```

exmc_pccard_disable

The description of exmc_pccard_disable is shown as below:

Table 3-391. Function exmc_pccard_disable

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

Example:

```
/* disable EXMC PC card bank */
exmc_pccard_disable();
```

exmc_interrupt_enable

The description of exmc_interrupt_enable is shown as below:

Table 3-392. Function exmc_interrupt_enable

Function name	exmc_interrupt_enable
Function prototype	void exmc_interrupt_enable(uint32_t exmc_bank,uint32_t interrupt);
Function descriptions	enable EXMC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
exmc_bank	specifies the NAND bank , PC card bank
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCA	the PC Card bank
RD	
Input parameter{in}	
interrupt	specify get which interrupt flag
EXMC_NAND_PCCAR_D_INT_FLAG_RISE	rising edge interrupt and corresponding flag
EXMC_NAND_PCCAR_D_INT_FLAG_LEVEL	high-level interrupt and corresponding flag
EXMC_NAND_PCCAR	falling edge interrupt and corresponding flag

<i>D_INT_FLAG_FALL</i>	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable EXMC rising edge interrupt*/
exmc_interrupt_enable(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

exmc_interrupt_disable

The description of `exmc_interrupt_disable` is shown as below:

Table 3-393. Function `exmc_interrupt_disable`

Function name	<code>exmc_interrupt_disable</code>
Function prototype	<code>void exmc_interrupt_disable(uint32_t exmc_bank,uint32_t interrupt);</code>
Function descriptions	disable EXMC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
<i>exmc_bank</i>	specifies the NAND bank , PC card bank
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA</i> <i>RD</i>	the PC Card bank
Input parameter{in}	
<i>interrupt</i>	specify get which interrupt flag
<i>EXMC_NAND_PCCAR</i> <i>D_INT_FLAG_RISE</i>	rising edge interrupt and corresponding flag
<i>EXMC_NAND_PCCAR</i> <i>D_INT_FLAG_LEVEL</i>	high-level interrupt and corresponding flag
<i>EXMC_NAND_PCCAR</i> <i>D_INT_FLAG_FALL</i>	falling edge interrupt and corresponding flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable EXMC rising edge interrupt*/
exmc_interrupt_disable(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

exmc_flag_get

The description of exmc_flag_get is shown as below:

Table 3-394. Function exmc_flag_get

Function name	exmc_flag_get
Function prototype	FlagStatus exmc_flag_get(uint32_t exmc_bank,uint32_t flag);
Function descriptions	get EXMC flag status
Precondition	-
The called functions	-
Input parameter{in}	
exmc_bank	specifies the NAND bank , PC card bank
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA RD</i>	the PC Card bank
Input parameter{in}	
flag	specify get which flag
<i>EXMC_NAND_PCCAR D_FLAG_RISE</i>	interrupt rising edge status
<i>EXMC_NAND_PCCAR D_FLAG_LEVEL</i>	interrupt high-level status
<i>EXMC_NAND_PCCAR D_FLAG_FALL</i>	interrupt falling edge status
<i>EXMC_NAND_PCCAR D_FLAG_FIFOE</i>	FIFO empty status
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check rising edge status is set or not*/
if(RESET != exmc_flag_get(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_FLAG_RISE));
```

exmc_flag_clear

The description of exmc_flag_clear is shown as below:

Table 3-395. Function exmc_flag_clear

Function name	exmc_flag_clear
Function prototype	FlagStatus exmc_flag_clear (uint32_t exmc_bank,uint32_t flag);
Function descriptions	clear EXMC flag status
Precondition	-

The called functions	-
Input parameter{in}	
exmc_bank	specifies the NAND bank , PC card bank
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA</i> <i>RD</i>	the PC Card bank
Input parameter{in}	
flag	specify get which flag
<i>EXMC_NAND_PCCAR</i> <i>D_FLAG_RISE</i>	interrupt rising edge status
<i>EXMC_NAND_PCCAR</i> <i>D_FLAG_LEVEL</i>	interrupt high-level status
<i>EXMC_NAND_PCCAR</i> <i>D_FLAG_FALL</i>	interrupt falling edge status
<i>EXMC_NAND_PCCAR</i> <i>D_FLAG_FIFOE</i>	FIFO empty status
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear rising edge status */
exmc_flag_clear(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_FLAG_RISE);
```

exmc_interrupt_flag_get

The description of `exmc_interrupt_flag_get` is shown as below:

Table 3-396. Function `exmc_interrupt_flag_get`

Function name	<code>exmc_interrupt_flag_get</code>
Function prototype	<code>FlagStatus exmc_interrupt_flag_get(uint32_t exmc_bank,uint32_t interrupt);</code>
Function descriptions	get EXMC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
exmc_bank	specifies the NAND bank , PC card bank
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA</i> <i>RD</i>	the PC Card bank
Input parameter{in}	

interrupt	specify get which interrupt flag
<i>EXMC_NAND_PCCAR_D_INT_FLAG_RISE</i>	rising edge interrupt and corresponding flag
<i>EXMC_NAND_PCCAR_D_INT_FLAG_LEVEL</i>	high-level interrupt and corresponding flag
<i>EXMC_NAND_PCCAR_D_INT_FLAG_FALL</i>	falling edge interrupt and corresponding flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check rising edge interrupt flag is set or not*/
if(RESET!=exmc_interrupt_flag_get(EXMC_BANK1_NAND,EXMC_NAND_PCCARD_INT_FLAG_RISE));
```

exmc_interrupt_flag_clear

The description of `exmc_interrupt_flag_clear` is shown as below:

Table 3-397. Function `exmc_interrupt_flag_clear`

Function name	<code>exmc_interrupt_flag_clear</code>
Function prototype	<code>void exmc_interrupt_flag_clear(uint32_t exmc_bank,uint32_t interrupt);</code>
Function descriptions	clear EXMC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
exmc_bank	specifies the NAND bank , PC card bank
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA_RD</i>	the PC Card bank
Input parameter{in}	
interrupt	specify get which interrupt flag
<i>EXMC_NAND_PCCAR_D_INT_FLAG_RISE</i>	rising edge interrupt and corresponding flag
<i>EXMC_NAND_PCCAR_D_INT_FLAG_LEVEL</i>	high-level interrupt and corresponding flag
<i>EXMC_NAND_PCCAR_D_INT_FLAG_FALL</i>	falling edge interrupt and corresponding flag
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* clear rising edge interrupt flag */

exmc_interrupt_flag_clear(EXMC_BANK1_NAND,EXMC_NAND_PCCARD_INT_FLAG_RI
SE);
```

3.13. EXTI

EXTI is the interrupt / event controller in the MCU. It contains up to 22 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 3-398. EXTI Registers

Registers	Descriptions
EXTI_INTEN	interrupt enable register
EXTI_EVENT	event enable register
EXTI_RTEN	rising edge trigger enable register
EXTI_FTEN	falling edge trigger enable register
EXTI_SWIEV	software interrupt event register
EXTI_PD	pending register

3.13.2. Descriptions of Peripheral functions

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

Table 3-399. 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

Function name	Function description
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 3-400. Enum exti_line_enum

Member name	Function description
EXTI_0	EXTI line 0
EXTI_1	EXTI line 1
EXTI_2	EXTI line 2
EXTI_3	EXTI line 3
EXTI_4	EXTI line 4
EXTI_5	EXTI line 5
EXTI_6	EXTI line 6
EXTI_7	EXTI line 7
EXTI_8	EXTI line 8
EXTI_9	EXTI line 9
EXTI_10	EXTI line 10
EXTI_11	EXTI line 11
EXTI_12	EXTI line 12
EXTI_13	EXTI line 13
EXTI_14	EXTI line 14
EXTI_15	EXTI line 15
EXTI_16	EXTI line 16
EXTI_17	EXTI line 17
EXTI_18	EXTI line 18
EXTI_19	EXTI line 19
EXTI_20	EXTI line 20
EXTI_21	EXTI line 21

Enum exti_mode_enum

Table 3-401. Enum exti_mode_enum

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

Enum exti_trig_type_enum

Table 3-402. 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 3-403. 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 3-404. 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 the EXTI
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-400. Enum exti_line_enum
Input parameter{in}	
mode	EXTI mode, refer to Table 3-401. Enum exti_mode_enum
Input parameter{in}	
trig_type	trigger type, refer to Table 3-402. 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 3-405. Function exti_interrupt_enable

Function name	enable the interrupts from EXTI line x
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 3-400. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the interrupts from EXTI line 0 */
exti_interrupt_enable(EXTI_0);
```

exti_interrupt_disable

The description of exti_interrupt_disable is shown as below:

Table 3-406. 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 3-400. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

exti_interrupt_disable(EXTI_0);
```

exti_event_enable

The description of exti_event_enable is shown as below:

Table 3-407. 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 3-400. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

exti_event_enable(EXTI_0);
```

exti_event_disable

The description of exti_event_disable is shown as below:

Table 3-408. 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 3-400. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
exti_event_disable(EXTI_0);
```

exti_software_interrupt_enable

The description of exti_software_interrupt_enable is shown as below:

Table 3-409. 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 EXTI software interrupt event
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-400. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable EXTI line 0 software interrupt event */
```

```
exti_software_interrupt_enable(EXTI_0);
```

exti_software_interrupt_disable

The description of exti_software_interrupt_disable is shown as below:

Table 3-410. 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 EXTI software interrupt event
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-400. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable EXTI line 0 software interrupt event */
```

```
exti_software_interrupt_disable(EXTI_0);
```

exti_flag_get

The description of exti_flag_get is shown as below:

Table 3-411. 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 3-400. Enum exti_line_enum	
Output parameter{out}		
-	-	
Return value		
FlagStatus	SET or RESET	

Example:

```
/* get EXTI line 0 flag status */
FlagStatus state = exti_flag_get(EXTI_0);
```

exti_flag_clear

The description of exti_flag_clear is shown as below:

Table 3-412. 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 3-400. 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 3-413. 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 3-400. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get EXTI line 0 interrupt flag status */
FlagStatus state = exti_interrupt_flag_get(EXTI_0);
```

exti_interrupt_flag_clear

The description of exti_interrupt_flag_clear is shown as below:

Table 3-414. 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 3-400. 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 GD32E50x 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 3-415. FMC Registers

Registers	Descriptions
FMC_WS	FMC wait state register
FMC_KEY	FMC unlock key register
FMC_OBKEY	FMC option bytes unlock key register
FMC_STAT	FMC status register
FMC_CTL	FMC control register
FMC_ADDR	FMC address register
FMC_OBSTAT	FMC option bytes status register
FMC_WP	FMC erase/program protection register
FMC_PID	FMC product ID register

3.14.2. Descriptions of Peripheral functions

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

Table 3-416. FMC firmware function

Function name	Function description
fmc_unlock	unlock the main FMC operation
fmc_lock	lock the main FMC operation
fmc_wscnt_set	set the FMC wait state
fmc_prefetch_enable	enable pre-fetch
fmc_prefetch_disable	disable pre-fetch
fmc_ibus_enable	enable IBUS cache
fmc_ibus_disable	disable IBUS cache
fmc_ibus_reset	reset IBUS cache
fmc_dbus_enable	enable DBUS cache
fmc_dbus_disable	disable DBUS cache
fmc_dbus_reset	reset DBUS cache
fmc_page_erase	FMC erase page
fmc_mass_erase	FMC erase whole chip
fmc_word_program	FMC program a word at the corresponding address
ob_unlock	unlock the option bytes operation
ob_lock	lock the option bytes operation

Function name	Function description
ob_erase	erase the option bytes
ob_write_protection_enable	enable write protection
ob_security_protection_config	configure the option bytes security protection
ob_user_write	program option bytes USER
ob_data_program	program option bytes DATA
ob_user_get	get the value of option bytes USER
ob_data_get	get the value of option bytes DATA
ob_write_protection_get	get the value of option bytes write protection
ob_security_protection_flag_get	get option bytes security protection state
fmc_flag_get	get FMC flag status
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 3-417. fmc_state_enum

enum name	enum description
FMC_READY	the operation has been completed
FMC_BUSY	the operation is in progress
FMC_PGERR	program error
FMC_PGAERR	program alignment error
FMC_WPERR	erase/program protection error
FMC_TOERR	timeout error
FMC_OB_HSPC	high security protection

fmc_unlock

The description of fmc_unlock is shown as below:

Table 3-418. Function fmc_unlock

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

-	-	-
---	---	---

Example:

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

```
fmc_unlock();
```

fmc_lock

The description of fmc_lock is shown as below:

Table 3-419. 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_wscnt_set

The description of fmc_wscnt_set is shown as below:

Table 3-420. Function fmc_wscnt_set

Function name	fmc_wscnt_set
Function prototype	void fmc_wscnt_set(uint32_t wscnt);
Function descriptions	set the wait state counter value
Precondition	-
The called functions	-
Input parameter{in}	
wscnt	wait state counter value
<i>FMC_WAIT_STATE_0</i>	0 wait state added
<i>FMC_WAIT_STATE_1</i>	1 wait state added
<i>FMC_WAIT_STATE_2</i>	2 wait state added
<i>FMC_WAIT_STATE_3</i>	3 wait state added
<i>FMC_WAIT_STATE_4</i>	4 wait state added

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set 1wait state */

fmc_wscnt_set(WS_WSCNT_1);
```

fmc_prefetch_enable

The description of fmc_prefetch_enable is shown as below:

Table 3-421. Function fmc_prefetch_enable

Function name	fmc_prefetch_enable
Function prototype	void fmc_prefetch_enable(void);
Function descriptions	enable pre-fetch
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable pre-fetch */

fmc_prefetch_enable();
```

fmc_prefetch_disable

The description of fmc_prefetch_disable is shown as below:

Table 3-422. Function fmc_prefetch_disable

Function name	fmc_prefetch_disable
Function prototype	void fmc_prefetch_disable (void);
Function descriptions	disable pre-fetch
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable pre-fetch */
fmc_prefetch_disable();
```

fmc_ibus_enable

The description of fmc_ibus_enable is shown as below:

Table 3-423. Function fmc_ibus_enable

Function name	fmc_ibus_enable
Function prototype	void fmc_ibus_enable(void);
Function descriptions	enable IBUS cache
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IBUS cache */
fmc_ibus_enable();
```

fmc_ibus_disable

The description of fmc_ibus_disable is shown as below:

Table 3-424. Function fmc_ibus_disable

Function name	fmc_ibus_disable
Function prototype	void fmc_ibus_disable(void);
Function descriptions	disable IBUS cache
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable IBUS cache */

fmc_ibus_disable();
```

fmc_ibus_reset

The description of fmc_ibus_reset is shown as below:

Table 3-425. Function fmc_ibus_reset

Function name	fmc_ibus_reset
Function prototype	void fmc_ibus_reset(void);
Function descriptions	reset IBUS cache
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset IBUS cache */

fmc_ibus_reset();
```

fmc_dbus_enable

The description of fmc_dbus_enable is shown as below:

Table 3-426. Function fmc_dbus_enable

Function name	fmc_dbus_enable
Function prototype	void fmc_dbus_enable(void);
Function descriptions	enable DBUS cache
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DBUS cache */
```

```
fmc_dbus_enable();
```

fmc_dbus_disable

The description of fmc_dbus_disable is shown as below:

Table 3-427. Function fmc_dbus_disable

Function name	fmc_dbus_disable
Function prototype	void fmc_dbus_disable(void);
Function descriptions	disable DBUS cache
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DBUS cache */
```

```
fmc_dbus_disable();
```

fmc_dbus_reset

The description of fmc_dbus_reset is shown as below:

Table 3-428. Function fmc_dbus_reset

Function name	fmc_dbus_reset
Function prototype	void fmc_dbus_reset(void);
Function descriptions	reset DBUS cache
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset DBUS cache */
```

```
fmc_dbus_reset();
```

fmc_page_erase

The description of fmc_page_erase is shown as below:

Table 3-429. Function fmc_page_erase

Function name	fmc_page_erase
Function prototype	fmc_state_enum fmc_page_erase(uint32_t page_address);
Function descriptions	FMC erase page
Precondition	fmc_unlock
The called functions	-
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 3-417. fmc_state_enum

Example:

```
/* erase page */
fmc_unlock();
fmc_state_enum state = fmc_page_erase(0x08004000);
```

fmc_mass_erase

The description of fmc_mass_erase is shown as below:

Table 3-430. Function fmc_mass_erase

Function name	fmc_mass_erase
Function prototype	fmc_state_enum fmc_mass_erase(void);
Function descriptions	FMC erase whole chip
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-417. fmc_state_enum

Example:

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

```

fmc_unlock();

fmc_state_enum state = fmc_mass_erase();
  
```

fmc_word_program

The description of fmc_word_program is shown as below:

Table 3-431. 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, fmc_page_erase / fmc_mass_erase
The called functions	-
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 3-417. fmc_state_enum

Example:

```

/* program a word at the corresponding address */

fmc_unlock();

fmc_page_erase(0x08004000);

fmc_state_enum state = fmc_word_program(0x08004000, 0xaabbccdd);
  
```

ob_unlock

The description of ob_unlock is shown as below:

Table 3-432. Function ob_unlock

Function name	ob_unlock
Function prototype	void ob_unlock(void);
Function descriptions	unlock the option bytes operation
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

Example:

```
/* unlock the option bytes operation */

fmc_unlock();

ob_unlock();
```

ob_lock

The description of ob_lock is shown as below:

Table 3-433. Function ob_lock

Function name	ob_lock
Function prototype	void ob_lock(void);
Function descriptions	lock the option bytes operation
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock the option bytes operation */

fmc_unlock();

ob_lock();
```

ob_erase

The description of ob_erase is shown as below:

Table 3-434. Function ob_erase

Function name	ob_erase
Function prototype	fmc_state_enum ob_erase(void);
Function descriptions	erase the FMC option bytes
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
fmc_state_enum	state of FMC, refer to Table 3-417. fmc_state_enum

Example:

```
/* erase the FMC option bytes */

fmc_state enum state;

fmc_unlock();

ob_unlock();

state = ob_erase();
```

ob_write_protection_enable

The description of ob_write_protection_enable is shown as below:

Table 3-435. 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	-
Input parameter{in}	
ob_wp	enable write protection
OB_WP_NONE	disable all write protection
OB_WPx	write protect specify sector x
OB_WP_ALL	write protect all sector
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-417. fmc_state_enum

Example:

```
/* enable write protection */

fmc_state enum state;

fmc_unlock();

ob_unlock();

state = ob_write_protection_enable(OB_WP7);
```

ob_security_protection_config

The description of ob_security_protection_config is shown as below:

Table 3-436. 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 security protection
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
ob_spc	specify security protection
FMC_NSPC	no security protection
FMC_LSPC	low security protection
FMC_HSPC	high security protection
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-417. fmc state enum

Example:

```
/* enable low security protection */

fmc_state enum state;

ob_unlock();

state = ob_security_protection_config(FMC_LSPC);
```

ob_user_write

The description of ob_user_write is shown as below:

Table 3-437. 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_stby, uint8_t ob_boot);
Function descriptions	program option bytes USER
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
ob_fwdgt	option bytes watchdog value
OB_FWDGT_SW	software free watchdog
OB_FWDGT_HW	hardware free watchdog
Input parameter{in}	
ob_deepsleep	option bytes deepsleep reset value
OB_DEEPSLEEP_NRS _T	no reset when entering deepsleep mode

OB_DEEPSLEEP_RST	generate a reset instead of entering deepsleep mode
Input parameter{in}	
ob_stdby	option bytes standby reset value
OB_STDBY_NRST	no reset when entering standby mode
OB_STDBY_RST	generate a reset instead of entering standby mode
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-417. fmc_state_enum

Example:

```
/* configure user option byte */

fmc_state_enum state;

fmc_unlock();

ob_unlock();

state = ob_user_write(OB_FWDGT_HW,OB_DEEPSLEEP_RST,
OB_STDBY_RST);
```

ob_data_program

The description of ob_data_program is shown as below:

Table 3-438. Function ob_data_program

Function name	ob_data_program
Function prototype	fmc_state_enum ob_data_program(uint16_t ob_data);
Function descriptions	program option bytes DATA
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
data	the byte to be programmed
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to Table 3-417. fmc_state_enum

Example:

```
/* program option bytes data */

fmc_state_enum state;

fmc_unlock();

ob_unlock();
```

```
state = ob_data_program(0x1ffff804, 0x56);
```

ob_user_get

The description of ob_user_get is shown as below:

Table 3-439. Function ob_user_get

Function name	ob_user_get
Function prototype	uint8_t ob_user_get(void);
Function descriptions	get the value of option bytes USER
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint8_t	the FMC user option bytes values(0x0 – 0xFF)

Example:

```
/* get the FMC user option bytes */

uint8_t user;

user = ob_user_get();
```

ob_data_get

The description of ob_data_get is shown as below:

Table 3-440. Function ob_data_program

Function name	ob_data_get
Function prototype	uint16_t ob_data_get(void);
Function descriptions	get the value of option bytes DATA
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
Uint16_t	the FMC data option bytes values(0x0 – 0xFFFF)

Example:

```
/* get the FMC data option bytes */
```

```

  uint16_t data;
  data = ob_data_get();

```

ob_write_protection_get

The description of ob_write_protection_get is shown as below:

Table 3-441. Function ob_write_protection_get

Function name	ob_write_protection_get
Function prototype	uint32_t ob_write_protection_get(void);
Function descriptions	get the value of option bytes write protection
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the FMC write protection option bytes value(0x0–0xFFFFFFFF)

Example:

```

/* get the FMC option bytes write protection */
uint32_t wp;
wp = ob_write_protection_get();

```

ob_security_protection_flag_get

The description of ob_security_protection_flag_get is shown as below:

Table 3-442. Function ob_security_protection_flag_get

Function name	ob_security_protection_flag_get
Function prototype	FlagStatus ob_security_protection_flag_get(void);
Function descriptions	get the FMC option bytes security protection state
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the FMC option bytes security protection */
```

```
FlagStatus flag;
```

```
flag = ob_security_protection_flag_get();
```

fmc_flag_get

The description of fmc_flag_get is shown as below:

Table 3-443. Function fmc_flag_get

Function name	fmc_flag_get
Function prototype	FlagStatus fmc_flag_get(uint32_t flag);
Function descriptions	get FMC flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	FMC flag
FMC_FLAG_BUSY	FMC busy flag
FMC_FLAG_PGERR	FMC program error flag
FMC_FLAG_PGAERR	FMC program alignment error flag
FMC_FLAG_WPERR	FMC erase/program protection error flag
FMC_FLAG_END	FMC end of operation flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get FMC end of operation flag */
```

```
FlagStatus flag;
```

```
flag = fmc_flag_get(FMC_FLAG_END);
```

fmc_flag_clear

The description of fmc_flag_clear is shown as below:

Table 3-444. Function fmc_flag_clear

Function name	fmc_flag_clear
Function prototype	void fmc_flag_clear(uint32_t flag);
Function descriptions	clear the FMC flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	FMC flag

<i>FMC_FLAG_PGERR</i>	FMC program error flag
<i>FMC_FLAG_PGAERR</i>	FMC program alignment error flag
<i>FMC_FLAG_WPERR</i>	FMC erase/program protection error flag
<i>FMC_FLAG_END</i>	FMC end of operation flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear FMC program error flag */
fmc_flag_clear(FMC_FLAG_PGERR);
```

fmc_interrupt_enable

The description of `fmc_interrupt_enable` is shown as below:

Table 3-445. Function `fmc_interrupt_enable`

Function name	<code>fmc_interrupt_enable</code>
Function prototype	<code>void fmc_interrupt_enable(uint32_t interrupt);</code>
Function descriptions	enable FMC interrupt
Precondition	<code>fmc_unlock</code>
The called functions	-
Input parameter{in}	
interrupt	FMC interrupt source
<i>FMC_INT_END</i>	FMC end of operation interrupt
<i>FMC_INT_ERR</i>	FMC error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable FMC end of operation interrupt */
fmc_unlock();
fmc_interrupt_enable(FMC_INT_END);
```

fmc_interrupt_disable

The description of `fmc_interrupt_disable` is shown as below:

Table 3-446. Function `fmc_interrupt_disable`

Function name	<code>fmc_interrupt_disable</code>
----------------------	------------------------------------

Function prototype	void fmc_interrupt_disable(uint32_t interrupt);
Function descriptions	disable FMC interrupt
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
interrupt	FMC interrupt source
<i>FMC_INT_END</i>	FMC end of operation interrupt
<i>FMC_INT_ERR</i>	FMC error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable FMC end of operation interrupt */

fmc_unlock();

fmc_interrupt_disable(FMC_INT_END);
```

fmc_interrupt_flag_get

The description of fmc_interrupt_flag_get is shown as below:

Table 3-447. Function fmc_interrupt_flag_get

Function name	fmc_interrupt_flag_get
Function prototype	FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum flag);
Function descriptions	get FMC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	FMC interrupt flag
<i>FMC_INT_FLAG_PGE</i> <i>RR</i>	FMC operation error interrupt flag
<i>FMC_INT_FLAG_PGA</i> <i>ERR</i>	FMC program alignment error interrupt flag
<i>FMC_INT_FLAG_WPE</i> <i>RR</i>	FMC erase/program protection error interrupt flag
<i>FMC_INT_FLAG_END</i> <i>K0_PGERR</i>	FMC end of operation interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```

/* get FMC operation error interrupt flag bit */

FlagStatus flag;

flag = fmc_interrupt_flag_get (FMC_INT_FLAG_PGERR);

fmc_interrupt_flag_clear
  
```

The description of `fmc_interrupt_flag_get` is shown as below:

Table 3-448. Function `fmc_interrupt_flag_clear`

Function name	<code>fmc_interrupt_flag_clear</code>
Function prototype	<code>void fmc_interrupt_flag_clear (fmc_interrupt_flag_enum flag);</code>
Function descriptions	clear FMC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	FMC flag
<i>FMC_INT_FLAG_PGE RR</i>	FMC operation error interrupt flag
<i>FMC_INT_FLAG_PGA ERR</i>	FMC program alignment error interrupt flag
<i>FMC_INT_FLAG_WPE RR</i>	FMC erase/program protection error interrupt flag
<i>FMC_INT_FLAG_END K0_PGERR</i>	FMC end of operation interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* clear FMC operation error interrupt flag */

fmc_interrupt_flag_clear (FMC_INT_FLAG_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 3-449. 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 3-450. 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 FWDGT counter prescaler value
fwdgt_reload_value_config	configure the FWDGT counter reload value
fwdgt_config	configure counter reload value, and prescaler divider value
fwdgt_counter_reload	reload the counter of FWDGT
fwdgt_flag_get	get flag state of FWDGT

fwdgt_write_enable

The description of fwdgt_write_enable is shown as below:

Table 3-451. 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 3-452. 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 3-453. 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 3-454. 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 FWDGT counter prescaler value
Precondition	-
The called functions	-
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	
-	-

Example:

```
/* configure the FWDGT counter prescaler value */

fwdgt_prescaler_value_config (FWDGT_PSC_DIV8);
```

fwdgt_reload_value_config

The description of fwdgt_reload_value_config is shown as below:

Table 3-455. 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 FWDGT counter reload value
Precondition	-
The called functions	-
Input parameter{in}	
reload_value	specify reload value(0x0000 - 0x0FFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the FWDGT counter reload value */

fwdgt_reload_value_config(625);
```

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-
<i>FWDGT_PSC_DIV4</i>	FWDGT prescaler set to 4
<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16
<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32
<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64
<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* confiure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */

fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

fwdgt_counter_reload

The description of fwdgt_counter_reload is shown as below:

Table 3-457. 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_flag_get

The description of fwdgt_flag_get is shown as below:

Table 3-458. Function fwdgt_flag_get

Function name	fwdgt_flag_get
Function prototype	FlagStatus fwdgt_flag_get(uint16_t flag);
Function descriptions	get flag state of FWDGT
Precondition	-
The called functions	-
Input parameter{in}	
flag	flag to get
<i>FWDGT_FLAG_PUD</i>	a write operation to FWDGT_PSC register is on going
<i>FWDGT_FLAG_RUD</i>	a write operation to FWDGT_RLD register is on going
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* test if a prescaler value update is on going */
```

```
FlagStatus status;

status = fwdgt_flag_get (FWDGT_FLAG_PUD);

if(status == RESET)

{
  ...

}

else
{
  ...
```

...

}

3.16. 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 3-459. 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
GPIOx_BOP	GPIO port bit operate register
GPIOx_BC	GPIO port bit clear register
GPIOx_LOCK	GPIO port configuration lock register
GPIOx_SPD	GPIO port bit speed register
AFIO_EC	AFIO event control register
AFIO_PCF0	AFIO port configuration register 0
AFIO_EXTI0	AFIO port EXTI sources selection register 0
AFIO_EXTI1	AFIO port EXTI sources selection register 1
AFIO_EXTI2	AFIO port EXTI sources selection register 2
AFIO_EXTI3	AFIO port EXTI sources selection register 3
AFIO_PCF1	AFIO port AFIO port configuration register 1
AFIO_CPSCTL	IO compensation control register
AFIO_PCFA	AFIO port configuration register A
AFIO_PCFB	AFIO port configuration register B
AFIO_PCFC	AFIO port configuration register C
AFIO_PCFD	AFIO port configuration register D
AFIO_PCFE	AFIO port configuration register E
AFIO_PCFG	AFIO port configuration register G

3.16.2. Descriptions of Peripheral functions

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

Table 3-460. 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_afio_port_config	configure AFIO port alternate function
gpio_ethernet_phy_select	select ethernet MII or RMII PHY (for GD32E50X_CL devices)
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
gpio_compensation_config	configure the I/O compensation cell
gpio_compensation_flag_get	check the I/O compensation cell is ready or not

gpio_deinit

The description of gpio_deinit is shown as below:

Table 3-461. Function gpio_deinit

Function name	gpio_deinit
Function prototype	void gpio_deinit(uint32_t gpio_periph);
Function descriptions	reset GPIO port
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
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 3-462. 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 3-463. 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
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
Input parameter{in}	
gpio_mode	gpio pin mode
GPIO_MODE_AIN	analog input mode
GPIO_MODE_IN_FLOATING	floating input mode
GPIO_MODE_IPD	pull-down input mode
GPIO_MODE_IPU	pull-up input mode

<code>GPIO_MODE_OUT_OD</code>	GPIO output with open-drain
<code>GPIO_MODE_OUT_PP</code>	GPIO output with push-pull
<code>GPIO_MODE_AF_OD</code>	AFIO output with open-drain
<code>GPIO_MODE_AF_PP</code>	AFIO output with push-pull
Input parameter{in}	
<code>speed</code>	gpio output max speed value
<code>GPIO_OSPEED_10MHZ</code>	output max speed 10MHz
<code>GPIO_OSPEED_2MHZ</code>	output max speed 2MHz
<code>GPIO_OSPEED_50MHZ</code>	output max speed 50MHz
<code>GPIO_OSPEED_MAX</code>	output max speed more than 50MHz
Input parameter{in}	
<code>pin</code>	GPIO pin
<code>GPIO_PIN_x</code>	<code>GPIO_PIN_x(x=0..15)</code>
<code>GPIO_PIN_ALL</code>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config 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 3-464. Function `gpio_bit_set`

Function name	<code>gpio_bit_set</code>
Function prototype	<code>void gpio_bit_set(uint32_t gpio_periph,uint32_t pin);</code>
Function descriptions	set GPIO pin
Precondition	-
The called functions	-
Input parameter{in}	
<code>gpio_periph</code>	GPIO port
<code>GPIOx</code>	<code>GPIOx (x=A,B,C,D,E,F,G)</code>
Input parameter{in}	
<code>pin</code>	GPIO pin
<code>GPIO_PIN_x</code>	<code>GPIO_PIN_x(x=0..15)</code>

GPIO_PIN_ALL	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 3-465. 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
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset PA0*/
gpio_bit_reset (GPIOA, GPIO_PIN_0);
```

gpio_bit_write

The description of gpio_bit_write is shown as below:

Table 3-466. Function gpio_bit_write

Function name	gpio_bit_write
Function prototype	void gpio_bit_write(uint32_t gpio_periph,uint32_t pin,bit_status bit_value);
Function descriptions	write data to the specified GPIO pin

Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	All pins
Input parameter{in}	
bit_value	SET or RESET
RESET	clear the port pin
SET	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 3-467. 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
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
Input parameter{in}	
data	specify the value to be written to the port output data register
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*write 1010 0101 1010 0101 to Port A */
```

```
gpio_port_write (GPIOA, 0xA5A5);
```

gpio_input_bit_get

The description of gpio_input_bit_get is shown as below:

Table 3-468. Function gpio_input_bit_get

Function name	gpio_input_bit_get
Function prototype	FlagStatus gpio_input_bit_get(uint32_t gpio_periph,uint32_t pin);
Function descriptions	get GPIO pin input status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get status of PA0 */
FlagStatus bit_state;
bit_state = gpio_input_bit_get(GPIOA, GPIO_PIN_0);
```

gpio_input_port_get

The description of gpio_input_port_get is shown as below:

Table 3-469. Function gpio_input_port_get

Function name	gpio_input_port_get
Function prototype	uint16_t gpio_input_port_get(uint32_t gpio_periph);
Function descriptions	get GPIO port input status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x=A,B,C,D,E,F,G)

Output parameter{out}	
-	-
Return value	
uint16_t	0x0000-0xFFFF

Example:

```
/* get input value of Port A */

uint16_t port_state;

port_state = gpio_input_port_get(GPIOA);
```

gpio_output_bit_get

The description of gpio_output_bit_get is shown as below:

Table 3-470. Function gpio_output_bit_get

Function name	gpio_output_bit_get
Function prototype	FlagStatus gpio_output_bit_get(uint32_t gpio_periph,uint32_t pin);
Function descriptions	get GPIO pin output status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get output status of PA0 */

FlagStatus bit_state;

bit_state = gpio_output_bit_get (GPIOA, GPIO_PIN_0);
```

gpio_output_port_get

The description of gpio_output_port_get is shown as below:

Table 3-471. Function gpio_output_port_get

Function name	gpio_output_port_get
---------------	----------------------

Function prototype	uint16_t gpio_output_port_get(uint32_t gpio_periph);
Function descriptions	get GPIO port output status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
Output parameter{out}	
-	-
Return value	
Uint16_t	0x0000-0xFFFF

Example:

```
/* get output value of Port A */

uint16_t port_state;

port_state = gpio_output_port_get (GPIOA);
```

gpio_pin_remap_config

The description of gpio_pin_remap_config is shown as below:

Table 3-472. Function gpio_pin_remap_config

Function name	gpio_pin_remap_config
Function prototype	void gpio_pin_remap_config(uint32_t remap, ControlStatus newvalue);
Function descriptions	configure GPIO pin remap
Precondition	-
The called functions	-
Input parameter{in}	
gpio_remap	select the pin to remap
GPIO_SPI0_REMAP	SPI0 remapping
GPIO_I2C0_REMAP	I2C0 remapping
GPIO_USART0_REMA P	USART0 remapping
GPIO_USART1_REMA P	USART1 remapping
GPIO_USART2_PARTI AL_REMAP	USART2 partial remapping
GPIO_USART2_FULL_ REMAP	USART2 full remapping
GPIO_TIMER0_PARTI AL_REMAP	TIMER0 partial remapping
GPIO_TIMER0_FULL_ REMAP	TIMER0 full remapping

<code>GPIO_TIMER1_PARTI_AL_REMAP0</code>	TIMER1 partial remapping
<code>GPIO_TIMER1_PARTI_AL_REMAP1</code>	TIMER1 partial remapping
<code>GPIO_TIMER1_FULL_REMAP</code>	TIMER1 full remapping
<code>GPIO_TIMER2_PARTI_AL_REMAP</code>	TIMER2 partial remapping
<code>GPIO_TIMER2_FULL_REMAP</code>	TIMER2 full remapping
<code>GPIO_TIMER3_REMAP</code>	TIMER3 remapping
<code>GPIO_PD01_REMAP</code>	PD01 remapping
<code>GPIO_TIMER4CH3_IR_EMAP</code>	TIMER4 channel3 internal remapping
<code>GPIO_ADC0_ETRGRT_REMAP</code>	ADC0 external trigger routine conversion remapping(only for GD32E50X_HD devices)
<code>GPIO_ADC1_ETRGRT_REMAP</code>	ADC1 external trigger routine conversion remapping(only for GD32E50X_HD devices)
<code>GPIO_ENET_REMAP</code>	ENET remapping(only for GD32E50X_CL devices)
<code>GPIO_SWJ_NONJTRS_T_REMAP</code>	full SWJ(JTAG-DP + SW-DP),but without NJTRST
<code>GPIO_SWJ_SWDPENABLE_REMAP</code>	JTAG-DP disabled and SW-DP enabled
<code>GPIO_SWJ_DISABLE_REMAP</code>	JTAG-DP disabled and SW-DP disabled
<code>GPIO_SPI2_REMAP</code>	SPI2 remapping
<code>GPIO_TIMER1ITR0_REMAP</code>	TIMER1 internal trigger 0 remapping(only for GD32E50X_CL devices)
<code>GPIO_PTP_PPS_REMAP</code>	ethernet PTP PPS remapping(only for GD32E50X_CL devices)
<code>GPIO_TIMER8_REMAP</code>	TIMER8 remapping
<code>GPIO_TIMER9_REMAP</code>	TIMER9 remapping
<code>GPIO_TIMER10_REMAP</code>	TIMER10 remapping
<code>GPIO_TIMER12_REMAP</code>	TIMER12 remapping
<code>GPIO_TIMER13_REMAP</code>	TIMER13 remapping
<code>GPIO_EXMC_NADV_REMAP</code>	EXMC_NADV connect/disconnect

<i>GPIO_CTC_REMAP0</i>	CTC remapping(PD15)
<i>GPIO_CTC_REMAP1</i>	CTC remapping(PF0)
Input parameter{in}	
<i>newvalue</i>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*enable SPI0 remapping*/
gpio_pin_remap_config (GPIO_SPI0_REMAP, ENABLE);
```

gpio_afio_port_config

The description of `gpio_afio_port_config` is shown as below:

Table 3-473. Function `gpio_afio_port_config`

Function name	gpio_afio_port_config
Function prototype	void gpio_afio_port_config(uint32_t afio_function, ControlStatus newvalue);
Function descriptions	configure AFIO port alternate function
Precondition	-
The called functions	-
Input parameter{in}	
<i>afio_function</i>	configure the port alternate function(SHRTIMER not support on GD32E50X_EPRT devices)
<i>AFIO_PA2_CMP1_CFG</i>	configure PA2 alternate function to CMP1
<i>AFIO_PA3_USBHS_CFG</i>	configure PA3 alternate function to USBHS
<i>AFIO_PA5_USBHS_CFG</i>	configure PA5 alternate function to USBHS
<i>AFIO_PA8_I2C2CFG</i>	configure PA8 alternate function to I2C2
<i>AFIO_PA8_SHRTIMER_ER_CFG</i>	configure PA8 alternate function to SHRTIMER
<i>AFIO_PA9_I2C2CFG</i>	configure PA9 alternate function to I2C2
<i>AFIO_PA9_SHRTIMER_ER_CFG</i>	configure PA9 alternate function to SHRTIMER
<i>AFIO_PA10_CMP5_</i>	configure PA10 alternate function to CMP5

CFG	
AFIO_PA10_SHRTI_MER_CFG	configure PA10 alternate function to SHRTIMER
AFIO_PA11_USART5_CFG	configure PA11 alternate function to USART5
AFIO_PA11_SHRTI_MER_CFG	configure PA11 alternate function to SHRTIMER
AFIO_PA12_CMP1_CFG	configure PA12 alternate function to CMP1
AFIO_PA12_USART5_CFG	configure PA12 alternate function to USART5
AFIO_PA12_SHRTI_MER_CFG	configure PA12 alternate function to SHRTIMER
AFIO_PA15_SHRTI_MER_CFG	configure PA15 alternate function to SHRTIMER
AFIO_PB0_USBHS_CFG	configure PB0 alternate function to USBHS
AFIO_PB1_CMP3_CFG	configure PB1 alternate function to CMP3
AFIO_PB1_USBHS_CFG	configure PB1 alternate function to USBHS
AFIO_PB1_SHRTIMER_CFG	configure PB1 alternate function to SHRTIMER
AFIO_PB2_USBHS_CFG	configure PB2 alternate function to USBHS
AFIO_PB2_SHRTIMER_CFG	configure PB2 alternate function to SHRTIMER
AFIO_PB3_SHRTIMER_CFG	configure PB3 alternate function to SHRTIMER
AFIO_PB4_I2S2_CFG	configure PB4 alternate function to I2S2
AFIO_PB4_I2C2_CFG	configure PB4 alternate function to I2C2
AFIO_PB4_SHRTIMER_CFG	configure PB4 alternate function to SHRTIMER
AFIO_PB5_I2C2_CFG	configure PB5 alternate function to I2C2
AFIO_PB5_USBHS_CFG	configure PB5 alternate function to USBHS
AFIO_PB5_SHRTIMER_CFG	configure PB5 alternate function to SHRTIMER
AFIO_PB6_SHRTIMER_CFG	configure PB6 alternate function to SHRTIMER

AFIO_PB7_SHRTIM_ER_CFG	configure PB7 alternate function to SHRTIMER
AFIO_PB8_I2C2_CFG	configure PB8 alternate function to I2C2
AFIO_PB8_SHRTIM_ER_CFG	configure PB8 alternate function to SHRTIMER
AFIO_PB9_CMP1_CFG	configure PB9 alternate function to CMP1
AFIO_PB9_SHRTIM_ER_CFG	configure PB9 alternate function to SHRTIMER
AFIO_PB10_USBHS_CFG	configure PB10 alternate function to USBHS
AFIO_PB10_SHRTIMER_CFG	configure PB10 alternate function to SHRTIMER
AFIO_PB11_USBHS_CFG	configure PB11 alternate function to USBHS
AFIO_PB11_SHRTIMER_CFG	configure PB11 alternate function to SHRTIMER
AFIO_PB12_USBHS_CFG	configure PB12 alternate function to USBHS
AFIO_PB12_SHRTIMER_CFG	configure PB12 alternate function to SHRTIMER
AFIO_PB13_USBHS_CFG	configure PB13 alternate function to USBHS
AFIO_PB13_SHRTIMER_CFG	configure PB13 alternate function to SHRTIMER
AFIO_PB14_I2S1_CFG	configure PB14 alternate function to I2S1
AFIO_PB14_SHRTIMER_CFG	configure PB14 alternate function to SHRTIMER
AFIO_PB15_SHRTIMER_CFG	configure PB15 alternate function to SHRTIMER
AFIO_PC0_USBHS_CFG	configure PC0 alternate function to USBHS
AFIO_PC2_I2S1_CFG	configure PC2 alternate function to I2S1
AFIO_PC2_USBHS_CFG	configure PC2 alternate function to USBHS
AFIO_PC3_USBHS_CFG	configure PC3 alternate function to USBHS
AFIO_PC6_CMP5_CFG	configure PC6 alternate function to CMP5
AFIO_PC6_USART5	configure PC6 alternate function to USART5

CFG	
AFIO_PC6_SHRTIM_ER_CFG	configure PC6 alternate function to SHRTIMER
AFIO_PC7_USART5_CFG	configure PC7 alternate function to USART5
AFIO_PC7_SHRTIM_ER_CFG	configure PC7 alternate function to SHRTIMER
AFIO_PC8_USART5_CFG	configure PC8 alternate function to USART5
AFIO_PC8_SHRTIM_ER_CFG	configure PC8 alternate function to SHRTIMER
AFIO_PC9_I2C2CFG	configure PC9 alternate function to I2C2
AFIO_PC9_SHRTIM_ER_CFG	configure PC9 alternate function to SHRTIMER
AFIO_PC10_I2C2CFG	configure PC10 alternate function to I2C2
AFIO_PC11_I2S2CFG	configure PC11 alternate function to I2S2
AFIO_PC11_SHRTIMER_MER_CFG	configure PC11 alternate function to SHRTIMER
AFIO_PC12_SHRTIMER_MER_CFG	configure PC12 alternate function to SHRTIMER
AFIO_PD4_SHRTIM_ER_CFG	configure PD4 alternate function to SHRTIMER
AFIO_PD5_SHRTIM_ER_CFG	configure PD5 alternate function to SHRTIMER
AFIO_PE0_SHRTIM_ER_CFG	configure PE0 alternate function to SHRTIMER
AFIO_PE1_SHRTIM_ER_CFG	configure PE1 alternate function to SHRTIMER
AFIO_PE8_CMP1CFG	configure PE8 alternate function to CMP1
AFIO_PE9_CMP3CFG	configure PE9 alternate function to CMP3
AFIO_PE10_CMP5CFG	configure PE10 alternate function to CMP5
AFIO_PE11_CMP5CFG	configure PE11 alternate function to CMP5
AFIO_PE12_CMP3CFG	configure PE12 alternate function to CMP3
AFIO_PE13_CMP1CFG	configure PE13 alternate function to CMP1

AFIO_PG6_SHRTIM_ER_CFG	configure PG6 alternate function to SHRTIMER_C
AFIO_PG7_USART5_CFG	configure PG7 alternate function to USART5
AFIO_PG7_SHRTIM_ER_CFG	configure PG7 alternate function to SHRTIMER
AFIO_PG9_USART5_CFG	configure PG9 alternate function to USART5
AFIO_PG10_SHRTIMER_CFG	configure PG10 alternate function to SHRTIMER
AFIO_PG11_SHRTIMER_CFG	configure PG11 alternate function to SHRTIMER
AFIO_PG12_SHRTIMER_CFG	configure PG12 alternate function to SHRTIMER
AFIO_PG13_SHRTIMER_CFG	configure PG13 alternate function to SHRTIMER
AFIO_PG14_USART5_CFG	configure PG14 alternate function to USART5
Input parameter{in}	
newvalue	control value
ENABLE	enable function
DISABLE	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure PG14 alternate function to USART5*/
gpio_afio_port_config (AFIO_PG14_USART5_CFG, ENABLE);
```

gpio_ethernet_phy_select

The description of gpio_ethernet_phy_select is shown as below:

Table 3-474. Function gpio_ethernet_phy_select

Function name	gpio_ethernet_phy_select
Function prototype	void gpio_ethernet_phy_select(uint32_t enet_sel);
Function descriptions	select ethernet MII or RMII PHY (for GD32E50X_CL devices)
Precondition	-
The called functions	-
Input parameter{in}	
enet_sel	ethernet MII or RMII PHY selection

GPIO_ENET_PHY_MII	configure ethernet MAC for connection with an MII PHY
GPIO_ENET_PHY_RMII	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 3-475. Function gpio_exti_source_select

Function name	gpio_exti_source_select
Function prototype	void gpio_exti_source_select(uint8_t output_port, uint8_t output_pin);
Function descriptions	select GPIO pin exti sources
Precondition	-
The called functions	-
Input parameter{in}	
output_port	gpio event output port
GPIO_PORT_SOURCE_GPIOf	output port source (x= A,B,C,D,E)
Input parameter{in}	
output_pin	gpio event output pin
GPIO_PIN_SOURCE_x	pin number (x=0..15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config 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 3-476. Function gpio_event_output_config

Function name	gpio_event_output_config
---------------	--------------------------

Function prototype	void gpio_event_output_config(uint8_t output_port, uint8_t output_pin);
Function descriptions	configure GPIO pin event output
Precondition	-
The called functions	-
Input parameter{in}	
output_port	gpio event output port
GPIO_EVENT_PORT_GPIOx	event output port x (x= A,B,C,D,E)
Input parameter{in}	
output_pin	gpio event output pin
GPIO_EVENT_PIN_x	pin number (x=0..15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* Config 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 3-477. Function gpio_event_output_enable

Function name	gpio_event_output_enable
Function prototype	void gpio_event_output_enable(void);
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 3-478. Function gpio_event_output_disable

Function name	gpio_event_output_disable
Function prototype	void gpio_event_output_disable(void);
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 3-479. 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
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* lock PA0 */

gpio_pin_lock (GPIOA, GPIO_PIN_0);

```

gpio_compensation_config

The description of gpio_compensation_config is shown as below:

Table 3-480. Function gpio_compensation_config

Function name	gpio_compensation_config
Function prototype	void gpio_compensation_config(uint32_t compensation);
Function descriptions	configure the I/O compensation cell
Precondition	-
The called functions	-
Input parameter{in}	
compensation	specifies the I/O compensation cell mode
GPIO_COMPENSATION_ENABLE	I/O compensation cell is enabled
GPIO_COMPENSATION_DISABLE	I/O compensation cell is disabled
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enabled I/O compensation cell */

gpio_compensation_config (GPIO_COMPENSATION_ENABLE);

```

gpio_compensation_flag_get

The description of gpio_compensation_flag_get is shown as below:

Table 3-481. Function gpio_compensation_flag_get

Function name	gpio_compensation_flag_get
Function prototype	FlagStatus gpio_compensation_flag_get(void);
Function descriptions	check the I/O compensation cell is ready or not
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```

/* check the I/O compensation cell state */

FlagStatus cell_state;

cell_state = gpio_compensation_flag_get (void);
  
```

3.17. SHRTIMER

SHRTIMER has a high-precision counting clock and can be used for high-precision timing. It can generate 10 high precision and flexible digital signals to control motor or be used for power management applications. The 10 digital signals can be output independently or coupled into 5 pairs of complementary signals. The SHRTIMER registers are listed in chapter [3.17.1](#), the SHRTIMER firmware functions are introduced in chapter [3.17.2](#). The GD32EPRT series does not have SHRTIMER module.

3.17.1. Descriptions of Peripheral registers

SHRTIMER registers are listed in the table shown as below:

Table 3-482. SHRTIMER Register

Registers	Descriptions
Master Timer Registers	
SHRTIMER_MTCTL0	Master_TIMER control register 0
SHRTIMER_MTINTF	Master_TIMER interrupt flag register
SHRTIMER_MTINTC	Master_TIMER interrupt flag clear register
SHRTIMER_MTDMAINTEN	Master_TIMER DMA and interrupt enable register
SHRTIMER_MTCNT	Master_TIMER counter register
SHRTIMER_MTCAR	Master_TIMER counter auto reload register
SHRTIMER_MTCREP	Master_TIMER counter repetition register
SHRTIMER_MTCMP0V	Master_TIMER compare 0 value register
SHRTIMER_MTCMP1V	Master_TIMER compare 1 value register
SHRTIMER_MTCMP2V	Master_TIMER compare 2 value register
SHRTIMER_MTCMP3V	Master_TIMER compare 3 value register
SHRTIMER_MTACTL	Master_TIMER additional control register
Slave Timer Registers	
SHRTIMER_STXCTL0	Slave_TIMERx control register 0
SHRTIMER_STXINTF	Slave_TIMERx interrupt flag register
SHRTIMER_STXINTC	Slave_TIMERx interrupt flag clear register
SHRTIMER_STXDMAINTEN	Slave_TIMERx DMA and interrupt enable register
SHRTIMER_STXCNT	Slave_TIMERx counter register
SHRTIMER_STXCAR	Slave_TIMERx counter auto reload register
SHRTIMER_STXCREP	Slave_TIMERx counter repetition register

Registers	Descriptions
SHRTIMER_STXCMP0V	Slave_TIMERx compare 0 value register
SHRTIMER_STXCMP0CP	Slave_TIMERx compare 0 composite register
SHRTIMER_STXCMP1V	Slave_TIMERx compare 1 value register
SHRTIMER_STXCMP2V	Slave_TIMERx compare 2 value register
SHRTIMER_STXCMP3V	Slave_TIMERx compare 3 value register
SHRTIMER_STXCAP0V	Slave_TIMERx capture 0 value register
SHRTIMER_STXCAP1V	Slave_TIMERx capture 1 value register
SHRTIMER_STXDTCTL	Slave_TIMERx dead-time control register
SHRTIMER_STXCH0SET	Slave_TIMERx channel 0 set request register
SHRTIMER_STXCH0RST	Slave_TIMERx channel 0 reset request register
SHRTIMER_STXCH1SET	Slave_TIMERx channel 1 set request register
SHRTIMER_STXCH1RST	Slave_TIMERx channel 1 reset request register
SHRTIMER_STXEXEVFCFG0	Slave_TIMERx external event filter configuration register 0
SHRTIMER_STXEXEVFCFG1	Slave_TIMERx external event filter configuration register 1
SHRTIMER_STXCNTRST	Slave_TIMERx counter reset register
SHRTIMER_STXCSCTL	Slave_TIMERx carrier-signal control register
SHRTIMER_STXCAP0TRG	Slave_TIMERx capture 0 trigger register
SHRTIMER_STXCAP1TRG	Slave_TIMERx capture 1 trigger register
SHRTIMER_STXCHOCTL	Slave_TIMERx channel output control register
SHRTIMER_STXFLTCTL	Slave_TIMERx fault control register
SHRTIMER_STXACTL	Slave_TIMERx additional control register
Common Registers	
SHRTIMER_CTL0	SHRTIMER control register 0
SHRTIMER_CTL1	SHRTIMER control register 1
SHRTIMER_INTF	SHRTIMER interrupt flag register
SHRTIMER_INTC	SHRTIMER interrupt flag clear register
SHRTIMER_INTEN	SHRTIMER interrupt enable register
SHRTIMER_CHOUTEN	SHRTIMER channel output enable register
SHRTIMER_CHOUTDIS	SHRTIMER channel output disable register
SHRTIMER_CHOUTDISF	SHRTIMER channel output disable flag register
SHRTIMER_BMCTL	SHRTIMER bunch mode control register
SHRTIMER_BMSTRG	SHRTIMER bunch mode start trigger register
SHRTIMER_BMCPV	SHRTIMER bunch mode compare value register
SHRTIMER_BMCAR	SHRTIMER bunch mode counter auto reload register
SHRTIMER_EXEVCFG0	SHRTIMER external event configuration register 0
SHRTIMER_EXEVCFG1	SHRTIMER external event configuration register 1
SHRTIMER_EXEVDFCTL	SHRTIMER external event digital filter control register
SHRTIMER_ADCTRIGS0	SHRTIMER trigger source 0 to ADC register
SHRTIMER_ADCTRIGS1	SHRTIMER trigger source 1 to ADC register

Registers	Descriptions
SHRTIMER_ADCTRIGS2	SHRTIMER trigger source 2 to ADC register
SHRTIMER_ADCTRIGS3	SHRTIMER trigger source 3 to ADC register
SHRTIMER_DLLCCTL	SHRTIMER DLL calibration control register
SHRTIMER_FLTINCFG0	SHRTIMER fault input configuration register 0
SHRTIMER_FLTINCFG1	SHRTIMER fault input configuration register 1
SHRTIMER_DMAUPMTR	SHRTIMER DMA update Master_TIMER register
SHRTIMER_DMAUPST0R	SHRTIMER DMA update Slave_TIMER0 register
SHRTIMER_DMAUPST1R	SHRTIMER DMA update Slave_TIMER1 register
SHRTIMER_DMAUPST2R	SHRTIMER DMA update Slave_TIMER2 register
SHRTIMER_DMAUPST3R	SHRTIMER DMA update Slave_TIMER3 register
SHRTIMER_DMAUPST4R	SHRTIMER DMA update Slave_TIMER4 register
SHRTIMER_DMATB	SHRTIMER DMA transfer buffer register

3.17.2. Descriptions of Peripheral functions

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

Table 3-483. SHRTIMER firmware function

Function name	Function description
shrtimer_deinit	deinit a SHRTIMER
shrtimer_dll_calibration_start	configure and start DLL calibration
shrtimer_baseinit_struct_para_init	initialize SHRTIMER time base parameters struct with a default value
shrtimer_timers_base_init	initialize Master_TIMER and Slave_TIMER timerbase
shrtimer_timers_counter_enable	enable a counter
shrtimer_timers_counter_disable	disable a counter
shrtimer_timers_update_event_enable	enable the Master_TIMER or Slave_TIMER update event
shrtimer_timers_update_event_disable	disable the Master_TIMER or Slave_TIMER update event
shrtimer_software_update	trigger the Master_TIMER and Slave_TIMER registers update by software
shrtimer_software_counter_reset	the Master_TIMER and Slave_TIMER counter reset by software
shrtimer_timerinit_struct_para_init	initialize waveform mode initialization parameters struct with a default value
shrtimer_timers_waveform_init	initialize a timer to work in waveform mode
shrtimer_timercfg_struct_para_init	initialize Slave_TIMER general behavior configuration struct with a default value
shrtimer_slavetimer_waveform_config	configure the general behavior of a Slave_TIMER which work in waveform mode
shrtimer_comparecfg_struct_para_init	initialize compare unit configuration struct with a default value

Function name	Function description
shrtimer_slavetimer_waveform_compare_config	configure the compare unit of a Slave_TIMER which work in waveform mode
shrtimer_channel_outputcfg_struct_para_init	initialize channel output configuration struct with a default value
shrtimer_slavetimer_waveform_channel_config	configure the channel output of a Slave_TIMER work in waveform mode
shrtimer_slavetimer_waveform_channel_software_request	software generates channel "set request" or "reset request"
shrtimer_slavetimer_waveform_channel_output_level_get	get Slave_TIMER channel output level
shrtimer_slavetimer_waveform_channel_state_get	get Slave_TIMER channel run state
shrtimer_deadtimercfg_struct_para_init	initialize dead time configuration struct with a default value
shrtimer_slavetimer_deadtime_config	configure the dead time for Slave_TIMER
shrtimer_carriersignalcfg_struct_para_init	initialize carrier signal configuration struct with a default value
shrtimer_slavetimer_carriersignal_config	configure the carrier signal mode for Slave_TIMER
shrtimer_output_channel_enable	enable a output channel
shrtimer_output_channel_disable	disable a output channel
shrtimer_mastertimer_compare_value_config	configure the compare value in Master_TIMER
shrtimer_mastertimer_compare_value_get	get the compare value in Master_TIMER
shrtimer_slavetimer_compare_value_config	configure the compare value in Slave_TIMER
shrtimer_slavetimer_compare_value_get	get the compare value in Slave_TIMER
shrtimer_timers_counter_value_config	configure the counter value in Master_TIMER and Slave_TIMER
shrtimer_timers_counter_value_get	get the counter value in Master_TIMER and Slave_TIMER
shrtimer_timers_autoreload_value_config	configure the counter auto reload value in Master_TIMER and Slave_TIMER
shrtimer_timers_autoreload_value_get	get the counter auto reload value in Master_TIMER and Slave_TIMER
shrtimer_timers_repetition_value_config	configure the counter repetition value in Master_TIMER and Slave_TIMER
shrtimer_timers_repetition_value_get	get the counter repetition value in Master_TIMER and Slave_TIMER
shrtimer_exevfilter_struct_para_init	initialize external event filtering for Slave_TIMER

Function name	Function description
	configuration struct with a default value
shrtimer_slavetimer_exevent_filtering_config	configure the external event filtering for Slave_TIMER (blanking, windowing)
shrtimer_exeventcfg_struct_para_init	initialize external event configuration struct with a default value
shrtimer_exevent_config	configure the an external event
shrtimer_exevent_prescaler	configure external event digital filter clock division
shrtimer_synccfg_struct_para_init	initialize synchronization configuration struct with a default value
shrtimer_synchronization_config	configure the synchronization input/output of the SHRTIMER
shrtimer_faultcfg_struct_para_init	configure the synchronization input/output of the SHRTIMER
shrtimer_fault_config	configure the fault input
shrtimer_fault_prescaler_config	configure the fault input digital filter clock division
shrtimer_fault_input_enable	fault input enable
shrtimer_fault_input_disable	fault input disable
shrtimer_timers_dma_enable	enable the Master_TIMER and Slave_TIMER DMA request
shrtimer_timers_dma_disable	disable the Master_TIMER and Slave_TIMER DMA request
shrtimer_dmamode_config	configure the DMA mode for Master_TIMER or Slave_TIMER
shrtimer_bunchmode_struct_para_init	initialize bunch mode configuration struct with a default value
shrtimer_bunchmode_config	configure bunch mode for the SHRTIMER
shrtimer_bunchmode_enable	enable the bunch mode
shrtimer_bunchmode_disable	disable the bunch mode
shrtimer_bunchmode_flag_get	get bunch mode operating flag
shrtimer_bunchmode_software_start	bunch mode started by software
shrtimer_slavetimer_capture_config	configure the capture source in Slave_TIMER
shrtimer_slavetimer_capture_software	capture triggered by software in Slave_TIMER
shrtimer_slavetimer_capture_value_read	read the capture value
shrtimer_adctrigcfg_struct_para_init	initialize ADC trigger configuration struct with a default value
shrtimer_adc_trigger_config	configure the trigger source to ADC and the update source
shrtimer_timers_flag_get	get the Master_TIMER and Slave_TIMER flag
shrtimer_timers_flag_clear	clear the Master_TIMER and Slave_TIMER flag
shrtimer_common_flag_get	get the common flag
shrtimer_common_flag_clear	clear the common flag
shrtimer_timers_interrupt_enable	enable the Master_TIMER and Slave_TIMER interrupt
shrtimer_timers_interrupt_disable	disable the Master_TIMER and Slave_TIMER interrupt
shrtimer_timers_interrupt_flag_get	clear the Master_TIMER and Slave_TIMER interrupt flag
shrtimer_timers_interrupt_flag_clear	clear the Master_TIMER and Slave_TIMER interrupt flag
shrtimer_common_interrupt_enable	enable the common interrupt
shrtimer_common_interrupt_disable	disable common interrupt
shrtimer_common_interrupt_flag_get	clear the common interrupt flag

Function name	Function description
shrtimer_common_interrupt_flag_clear	clear the common interrupt flag

Structure shrtimer_baseinit_parameter_struct

Table 3-484. Structure shrtimer_baseinit_parameter_struct

member name	Function description
period	period value, min value: 3 tSHRTIMER_CK clock, max value: 0xFFFF - (1 tSHRTIMER_CK)
repetitioncounter	the counter repetition value, 0x00~0xFF
prescaler	prescaler value, refer to: counter clock division
counter_mode	counter operating mode, refer to: counter operating mode

Structure shrtimer_timerinit_parameter_struct

Table 3-485. Structure shrtimer_timerinit_parameter_struct

member name	Function description
half_mode	specifies whether or not half mode is enabled, refer to: half mode enabling status
start_sync	specifies whether or not timer is started by a rising edge on the synchronization input, refer to: synchronous input start timer
reset_sync	specifies whether or not timer is reset by a rising edge on the synchronization input, refer to: synchronous input reset timer
dac_trigger	indicates whether or not the a DAC synchronization event is generated, refer to: trigger source to DAC
shadow	Indicates whether or not the a DAC synchronization event is generated, refer to: trigger source to DAC
update_selection	the update occurs with respect to DMA mode or STxUPINy (Slave_TIMERx only), refer to: update event selection
cnt_bunch	the timer behaves during a bunch mode operation, refer to: timer behaves during a bunch mode operation
repetition_update	the timer behaves during a bunch mode operation, refer to: timer behaves during a bunch mode operation

Structure shrtimer_timercfg_parameter_struct

Table 3-486. Structure shrtimer_timercfg_parameter_struct

member name	Function description
balanced_mode	specifies whether or not the balanced mode is enabled, refer to: set balanced mode
fault_enable	specifies whether or not the fault channels are enabled for the Slave_TIMER, refer to: fault channel enabled for a Slave_TIMER
fault_protect	specifies whether the write protection function is enable or not, refer to: protect

	fault enable
deadtime_enable	specifies whether or not dead time insertion is enabled for the timer, refer to: dead time enable
delayed_idle	the delayed IDLE mode, refer to: set delayed IDLE state mode
update_source	the source triggering the Slave_TIMER registers update, refer to: update is done synchronously with any other Slave_TIMER or Master_TIMER update
cnt_reset	the source triggering the Slave_TIMER counter reset, refer to: Slave_TIMER counter reset
reset_update	specifies whether or not registers update is triggered when the timer counter is reset, refer to: update event generated by reset event

Structure shrtimer_COMPAREcfg_parameter_struct

Table 3-487. Structure shrtimer_COMPAREcfg_parameter_struct

member name	Function description
compare_value	compare value, min value: 3 tSHRTIMER_CK clock, max value: 0xFFFF - (1 tSHRTIMER_CK)
delayed_mode	defining whether the compare register is behaving in regular mode or in delayed mode, refer to: compare 3 or 1 delayed mode
timeout_value	compare value for compare 0 or 2 when compare 3 or 1 is delayed mode with time out is selected , timeout_value + compare_value must be less than 0xFFFF

Structure shrtimer_EXEVFILTER_parameter_struct

Table 3-488. Structure shrtimer_EXEVFILTER_parameter_struct

member name	Function description
filter_mode	the external event filter mode for Slave_TIMER, refer to: external event filter mode
memorized	specifies whether or not the signal is memorized, refer to: external event memorized enable

Structure shrtimer_DEADTIMEcfg_parameter_struct

Table 3-489. Structure shrtimer_DEADTIMEcfg_parameter_struct

member name	Function description
prescaler	dead time generator clock division, refer to: dead time prescaler
rising_value	rising edge dead-time value, 0x0000~0xFFFF
rising_sign	the sign of rising edge dead-time value, refer to: dead time rising sign
rising_protect	dead time rising edge protection for value and sign, refer to: dead time rising edge protection for value and sign
risingsign_protect	dead time rising edge protection for sign, refer to: dead time rising edge protection only for sign
falling_value	falling edge dead-time value, 0x0000~0xFFFF

falling_sign	the sign of falling edge dead-time value, refer to: dead time falling sign
falling_protect	dead time falling edge protection for value and sign, refer to: dead time falling edge protection for value and sign
fallingsign_protect	dead time falling edge protection for sign, refer to: dead time falling edge protection only for sign

Structure shrtimer_carriersignalcfg_parameter_struct

Table 3-490. Structure shrtimer_carriersignalcfg_parameter_struct

member name	Function description
period	carrier signal period: tCSPRD, 0x0~0xF. tCSPRD = (period + 1) * 16 * tSHRTIMER_CK
duty_cycle	carrier signal duty cycle, 0x0~0x7, duty cycle = duty_cycle/8
first_pulse	first carrier-signal pulse width: tCSFSTPW, 0x0~0xF. tCSFSTPW = (first_pulse+1) * 16 * tSHRTIMER_CK

Structure shrtimer_synccfg_parameter_struct

Table 3-491. Structure shrtimer_synccfg_parameter_struct

member name	Function description
input_source	the external synchronization input source, refer to: the synchronization input source
output_source	the source and event to be sent on the external synchronization outputs, refer to: the synchronization output source
output_polarity	the polarity and length of the pulse to be sent on the external synchronization outputs, refer to: the pulse on the synchronization output pad SHRTIMER_SCOUT

Structure shrtimer_bunchmode_parameter_struct

Table 3-492. Structure shrtimer_bunchmode_parameter_struct

member name	Function description
mode	the bunch mode operating mode, refer to: continuous mode in bunch mode
clock_source	specifies the burst mode clock source, refer to: bunch mode clock source
prescaler	the bunch mode prescaler, refer to: bunch mode clock division
shadow	specifies whether or not preload is enabled for SHRTIMER_BMCMPV and SHRTIMER_BMCAR registers, refer to: bunch mode shadow enable
trigger	the event triggering the bunch operation, refer to: the event triggers bunch mode operation
idle_duration	the duration of the IDLE, 0x0000~0xFFFF
period	the bunch mode period which is the sum of the IDLE and RUN duration, 0x0001~0xFFFF

Structure shrtimer_exeventcfg_parameter_struct

Table 3-493. Structure shrtimer_exeventcfg_parameter_struct

member name	Function description
source	the source of the external event, refer to: external event source
polarity	the active level of external event 0 when EXEVyEG[1:0] = 2'b00, refer to: external event polarity
edge	the sensitivity of the external event, external event edge sensitivity
digital_filter	external event filter control, 0x0~0xF

Structure shrtimer_faultcfg_parameter_struct

Table 3-494. Structure shrtimer_faultcfg_parameter_struct

member name	Function description
source	the source of the fault input, refer to: fault input source
polarity	the polarity of the fault input, refer to: fault input polarity
filter	fault input filter control, 0x0~0xF
control	fault input enable or disable, refer to: enable or disable fault
protect	protect fault input configuration, refer to: protect fault input configuration

Structure shrtimer_adctrigcfg_parameter_struct

Table 3-495. Structure shrtimer_adctrigcfg_parameter_struct

member name	Function description
update_source	the source triggering the update of the SHRTIMER_ADCTRIGSy register, refer to: SHRTIMER_ADCTRIG update source
trigger	the event triggering the ADC conversion, refer to: ADC trigger event

Structure shrtimer_channel_outputcfg_parameter_struct

Table 3-496. Structure shrtimer_channel_outputcfg_parameter_struct

member name	Function description
polarity	configure channel output polarity, refer to: channel output polarity
set_request	configure the event generates channel 'set request', refer to channel set request
reset_request	configure the event generates channel 'reset request', refer to: channel reset request
idle_bunch	specifies whether channel output can be IDLE state in bunch mode, refer to: channel IDLE state enable in bunch mode
idle_state	specifies channel output idle state, refer to channel output idle state
fault_state	specifies the output level when in FAULT state, refer to: channel output in fault state
carrier_mode	specifies whether or not the carrier-signal mode is enabled, refer to: channel

	carrier-signal mode enable
deadtime_bunch	specifies whether or not deadtime is inserted before output entering the IDLE state in bunch mode, refer to: channel dead-time insert in bunch mode

shrtimer_deinit

The description of shrtimer_deinit is shown as below:

Table 3-497. Function shrtimer_deinit

Function name	shrtimer_deinit
Function prototype	void shrtimer_deinit(uint32_t shrtimer_periph);
Function descriptions	deinit SHRTIMER
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset SHRTIMER */

shrtimer_deinit(SHRTIMER0);
```

shrtimer_dll_calibration_start

The description of shrtimer_dll_calibration_start is shown as below:

Table 3-498. Function shrtimer_dll_calibration_start

Function name	shrtimer_dll_calibration_start
Function prototype	void shrtimer_dll_calibration_start(uint32_t shrtimer_periph);
Function descriptions	configure and start DLL calibration
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
calform	specify the calibration form
SHRTIMER_CALIBRATION_ONCE	DLL calibration start once
SHRTIMER_CALIBRATION	DLL periodic calibration, the length of the DLL calibration cycle is 1048576 *

<i>ION_1048576_PERIOD</i>	tSHRTIMER_CK
<i>SHRTIMER_CALIBRAT</i>	DLL periodic calibration, the length of the DLL calibration cycle is 131072 *
<i>ION_131072_PERIOD</i>	tSHRTIMER_CK
<i>SHRTIMER_CALIBRAT</i>	DLL periodic calibration, the length of the DLL calibration cycle is 16384 *
<i>ION_16384_PERIOD</i>	tSHRTIMER_CK
<i>SHRTIMER_CALIBRAT</i>	DLL periodic calibration, the length of the DLL calibration cycle is 2048 *
<i>ION_2048_PERIOD</i>	tSHRTIMER_CK
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure and start DLL calibration */

shrtimer_dll_calibration_start(SHRTIMER0, SHRTIMER_CALIBRATION_ONCE);
```

shrtimer_baseinit_struct_para_init

The description of shrtimer_baseinit_struct_para_init is shown as below:

Table 3-499. Function shrtimer_baseinit_struct_para_init

Function name	shrtimer_baseinit_struct_para_init
Function prototype	void shrtimer_baseinit_struct_para_init(shrtimer_baseinit_parameter_struct* baseinit);
Function descriptions	initialize SHRTIMER time base parameters struct with the default value
Precondition	-
The called functions	-
Input parameter{in}	
baseinit	shrtimer_baseinit_parameter_struct, the structure members can refer to Table 3-484. Structure shrtimer_baseinit_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize SHRTIMER time base parameters struct with the default value */

shrtimer_baseinit_parameter_struct baseinit;

shrtimer_baseinit_struct_para_init(&baseinit);
```

shrtimer_timers_base_init

The description of shrtimer_timers_base_init is shown as below:

Table 3-500. Function shrtimer_timers_base_init

Function name	shrtimer_timers_base_init
Function prototype	void shrtimer_timers_base_init(uint32_t shrtimer_periph);
Function descriptions	initialize Master_TIMER and Slave_TIMER timerbase
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	master timer and slave timer index
SHRTIMER_MASTER_TIMER	the master timer
SHRTIMER_SLAVE_TIMERx	the slave timer selection(x=0..4)
Input parameter{in}	
baseinit	shrtimer_baseinit_parameter_struct, the structure members can refer to Table 3-484. Structure shrtimer_baseinit_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize Master_TIMER and Slave_TIMER timerbase */

shrtimer_baseinit_parameter_struct baseinit_para;

shrtimer_baseinit_struct_para_init(&baseinit_para);

baseinit_para.period = 384;

baseinit_para.prescaler = SHRTIMER_PRESCALER_MUL64;

baseinit_para.repetitioncounter = 0;

baseinit_para.counter_mode = SHRTIMER_COUNTER_MODE_CONTINOUS;

shrtimer_timers_base_init(SHRTIMER0, SHRTIMER_SLAVE_TIMER0, &baseinit_para);

```

shrtimer_timers_counter_enable

The description of shrtimer_timers_counter_enable is shown as below:

Table 3-501. Function shrtimer_timers_counter_enable

Function name	shrtimer_timers_counter_enable
Function prototype	void shrtimer_timers_counter_enable(uint32_t shrtimer_periph, uint32_t

	cntid);
Function descriptions	enable a counter
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
cntid	specify the counter to configure
SHRTIMER_MT_COUN TER	the counter of Master_TIMER
SHRTIMER_STx_COU NTER(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable a counter */

void shrtimer_timers_counter_enable(SHRTIMER0, SHRTIMER_MT_COUNTER);

shrtimer_timers_counter_disable
```

The description of shrtimer_timers_counter_disable is shown as below:

Table 3-502. Function shrtimer_timers_counter_enable

Function name	shrtimer_timers_counter_disable
Function prototype	void shrtimer_timers_counter_disable(uint32_t shrtimer_periph, uint32_t cntid);
Function descriptions	disable a counter
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
cntid	specify the counter to configure
SHRTIMER_MT_COUN TER	the counter of Master_TIMER
SHRTIMER_STx_COU NTER(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable a counter */

shrtimer_timers_counter_disable(SHRTIMER0, SHRTIMER_MT_COUNTER);
```

shrtimer_timers_update_event_enable

The description of shrtimer_timers_update_event_enable is shown as below:

Table 3-503. Function shrtimer_timers_update_event_enable

Function name	shrtimer_timers_update_event_enable
Function prototype	void shrtimer_timers_update_event_enable(uint32_t shrtimer_periph, uint32_t timer_id);
Function descriptions	enable the Master_TIMER or Slave_TIMER update event
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER peripheral
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the Master_TIMER or Slave_TIMER update event */

shrtimer_timers_update_event_enable(SHRTIMER0, SHRTIMER_MASTER_TIMER);
```

shrtimer_timers_update_event_disable

The description of shrtimer_timers_update_event_disable is shown as below:

Table 3-504. Function shrtimer_timers_update_event_disable

Function name	shrtimer_timers_update_event_disable
Function prototype	void shrtimer_timers_update_event_disable(uint32_t shrtimer_periph,

	uint32_t timer_id);
Function descriptions	disable the Master_TIMER or Slave_TIMER update event
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER peripheral
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the Master_TIMER or Slave_TIMER update event */

shrtimer_timers_update_event_disable(SHRTIMER0, SHRTIMER_MASTER_TIMER);
```

shrtimer_software_update

The description of shrtimer_software_update is shown as below:

Table 3-505. Function shrtimer_software_update

Function name	shrtimer_software_update
Function prototype	void shrtimer_software_update(uint32_t shrtimer_periph, uint32_t timersrc);
Function descriptions	update the Master_TIMER or Slave_TIMER by software
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timersrc	software update timer selection
SHRTIMER_UPDATE_SW_MT	Master_TIMER software update
SHRTIMER_UPDATE_SW_STx(x=0..4)	Slave_TIMERx software update (x=0..4)
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* update the Master_TIMER or Slave_TIMER by software */

void shrtimer_software_update(SHRTIMER0, SHRTIMER_UPDATE_SW_MT);
```

shrtimer_software_counter_reset

The description of shrtimer_software_counter_reset is shown as below:

Table 3-506. Function shrtimer_software_counter_reset

Function name	shrtimer_software_counter_reset
Function prototype	void shrtimer_software_counter_reset(uint32_t shrtimer_periph, uint32_t timerrst);
Function descriptions	reset the Master_TIMER or Slave_TIMER counter by software
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timersrc	software reset timer selection
SHRTIMER_COUNTER_RESET_SW_MT	Master_TIMER software reset
SHRTIMER_COUNTER_RESET_SW_STx (x=0..4)	Slave_TIMERx software reset(x=0..4)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* triggers the Master_TIMER or Slave_TIMER registers update by software */

void shrtimer_software_counter_reset(SHRTIMER0,
SHRTIMER_COUNTER_RESET_SW_MT);
```

shrtimer_timerinit_struct_para_init

The description of shrtimer_timerinit_struct_para_init is shown as below:

Table 3-507. Function shrtimer_timerinit_struct_para_init

Function name	shrtimer_timerinit_struct_para_init
----------------------	-------------------------------------

Function prototype	void shrtimer_timerinit_struct_para_init(shrtimer_timerinit_parameter_struct* timerinit);
Function descriptions	initialize waveform mode initialization parameters struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
timerinit	shrtimer_timerinit_parameter_struct, the structure members can refer to Table 3-485. Structure shrtimer_timerinit_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize waveform mode initialization parameters struct with a default value */

shrtimer_timerinit_parameter_struct timerinit_para;
shrtimer_struct_para_init(&timerinit_para);
```

shrtimer_timers_waveform_init

The description of shrtimer_timers_waveform_init is shown as below:

Table 3-508. Function shrtimer_timers_waveform_init

Function name	shrtimer_timers_waveform_init
Function prototype	void shrtimer_timers_waveform_init(uint32_t shrtimer_periph, uint32_t timer_id, shrtimer_timerinit_parameter_struct* timerinitpara);
Function descriptions	initialize a timer to work in waveform mode
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER peripheral
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
timerinit	shrtimer_timerinit_parameter_struct, the structure members can refer to Table 3-485. Structure shrtimer_timerinit_parameter_struct
Output parameter{out}	
-	-

Return value	
-	-

Example:

```

/* initialize a timer to work in waveform mode */

shrtimer_timerinit_parameter_struct timerinit_para;

shrtimer_timerinit_struct_para_init(&timerinit_para);

timerinit_para.cnt_bunch = SHRTIMER_TIMERBUNCHNMODE_MAINTAINCLOCK;

timerinit_para.DAC_trigger = SHRTIMER_DAC_TRIGGER_NONE;

timerinit_para.half_mode = SHRTIMER_HALFMODE_DISABLED;

timerinit_para.repetition_update = SHRTIMER_UPDATEONREPETITION_DISABLED;

timerinit_para.reset_sync = SHRTIMER_SYNCRESET_DISABLED;

timerinit_para.shadow = SHRTIMER_SHADOW_DISABLED;

timerinit_para.start_sync = SHRTIMER_SYNISTART_DISABLED;

timerinit_para.update_selection
SHRTIMER_MT_ST_UPDATE_SELECTION_INDEPENDENT;

shrtimer_timers_waveform_init(SHRTIMER0,           SHRTIMER_SLAVE_TIMER0,
&timerinit_para);

```

shrtimer_timercfg_struct_para_init

The description of shrtimer_timercfg_struct_para_init is shown as below:

Table 3-509. Function shrtimer_timercfg_struct_para_init

Function name	shrtimer_timercfg_struct_para_init
Function prototype	void shrtimer_timercfg_struct_para_init(shrtimer_timercfg_parameter_struct* timercfg);
Function descriptions	initialize Slave_TIMER general behavior configuration struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
timercfg	shrtimer_timercfg_parameter_struct, the structure members can refer to Table 3-486. Structure shrtimer_timercfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize Slave_TIMER general behavior configuration struct with a default value */

shrtimer_timercfg_parameter_struct timercfg_para;

shrtimer_timercfg_struct_para_init(timercfg_para);
```

shrtimer_slavetimer_waveform_config

The description of shrtimer_slavetimer_waveform_config is shown as below:

Table 3-510. Function shrtimer_slavetimer_waveform_config

Function name	shrtimer_slavetimer_waveform_config
Function prototype	void shrtimer_slavetimer_waveform_config(uint32_t shrtimer_periph, uint32_t timer_id, shrtimer_timercfg_parameter_struct * timercfg);
Function descriptions	initialize Slave_TIMER general behavior configuration struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER peripheral
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
timercfg	shrtimer_timercfg_parameter_struct, the structure members can refer to Table 3-486. Structure shrtimer_timercfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize Slave_TIMER general behavior configuration struct with a default value */

shrtimer_timercfg_parameter_struct timercfg_para;

shrtimer_timercfg_struct_para_init(&timercfg_para);

timercfg_para.balanced_mode = SHRTIMER_STXBALANCEDMODE_DISABLED;

timercfg_para.cnt_reset = SHRTIMER_STXCNT_RESET_NONE;
```

```

timercfg_para.deadtime_enable = SHRTIMER_STXDEADTIME_DISABLED;
timercfg_para.delayed_idle = SHRTIMER_STXDELAYED_IDLE_DISABLED;
timercfg_para.fault_enable = SHRTIMER_STXFAULTENABLE_NONE;
timercfg_para.fault_protect = SHRTIMER_STXFAULT_PROTECT_READWRITE;
timercfg_para.reset_update = SHRTIMER_STXUPDATEONRESET_DISABLED;
timercfg_para.update_source = SHRTIMER_STXUPDATETRIGGER_NONE;
shrtimer_slavetimer_waveform_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
&timercfg_para);
  
```

shrtimer_comparecfg_struct_para_init

The description of shrtimer_comparecfg_struct_para_init is shown as below:

Table 3-511. Function shrtimer_comparecfg_struct_para_init

Function name	shrtimer_comparecfg_struct_para_init
Function prototype	void shrtimer_comparecfg_struct_para_init(shrtimer_comparecfg_parameter_struct* comparecfg);
Function descriptions	initialize compare unit configuration struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
comparecfg	timer parameter initialization struct, the structure members can refer to Table 3-487. Structure shrtimer_comparecfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize compare unit configuration struct with a default value */

shrtimer_comparecfg_parameter_struct comparecfg_para;
shrtimer_comparecfg_struct_para_init(comparecfg_para);
  
```

shrtimer_slavetimer_waveform_compare_config

The description of shrtimer_slavetimer_waveform_compare_config is shown as below:

Table 3-512. Function shrtimer_slavetimer_waveform_compare_config

Function name	shrtimer_slavetimer_waveform_compare_config
Function prototype	void shrtimer_slavetimer_waveform_compare_config(uint32_t

	shrtimer_periph, uint32_t timer_id, uint32_t comparex, shrtimer_comparecfg_parameter_struct* cmpcfg);
Function descriptions	configure the compare unit of a Slave_TIMER which work in waveform mode
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER peripheral
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
comparex	compare unit selection
SHRTIMER_COMPAR_Ey (y=0..4)	the compare unit y(y=0..4)
Input parameter{in}	
comparecfg	timer parameter initialization struct, the structure members can refer to Table 3-487. Structure shrtimer_comparecfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the compare unit of a Slave_TIMER which work in waveform mode */

shrtimer_comparecfg_parameter_struct comparecfg_para;
shrtimer_comparecfg_struct_para_init(&comparecfg_para);
comparecfg_para.compare_value = 192;
shrtimer_slavetimer_waveform_compare_config(SHRTIMER0,
SHRTIMER_SLAVE_TIMER0, SHRTIMER_COMPARE0, &comparecfg_para);
```

shrtimer_channel_outputcfg_struct_para_init

The description of shrtimer_channel_outputcfg_struct_para_init is shown as below:

Table 3-513. Function shrtimer_channel_outputcfg_struct_para_init

Function name	shrtimer_channel_outputcfg_struct_para_init
Function prototype	void

	shrtimer_channel_outputcfg_struct_para_init(shrtimer_channel_outputcfg_parameter_struct * channelcfg);
Function descriptions	initialize channel output configuration struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
channelcfg	shrtimer_channel_outputcfg_parameter_struct, the structure members can refer to Table 3-496. Structure shrtimer_channel_outputcfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize channel output configuration struct with a default value */

shrtimer_channel_outputcfg_parameter_struct outcfg_para;
shrtimer_channel_outputcfg_struct_para_init(&outcfg_para);
```

shrtimer_slavetimer_waveform_channel_config

The description of shrtimer_slavetimer_waveform_channel_config is shown as below:

Table 3-514. Function shrtimer_slavetimer_waveform_channel_config

Function name	shrtimer_slavetimer_waveform_channel_config
Function prototype	void shrtimer_slavetimer_waveform_channel_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t channel, shrtimer_channel_outputcfg_parameter_struct * channelcfg);
Function descriptions	configure the channel of a Slave_TIMER work in waveform mode
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER timer index
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
channel	SHRTIMER timer channel index
SHRTIMER_STx_CHy()	SHRTIMER timer channel selection

x=0..4,y=0,1)	
Input parameter{in}	
channelcfg	shrtimer_channel_outputcfg_parameter_struct, the structure members can refer to Table 3-496. Structure shrtimer_channel_outputcfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure the channel of a Slave_TIMER work in waveform mode */

shrtimer_channel_outputcfg_parameter_struct outcfg_para;

shrtimer_channel_outputcfg_struct_para_init(&outcfg_para);

outcfg_para.carrier_mode = SHRTIMER_CHANNEL_CARRIER_DISABLED;

outcfg_para.deadtime_bunch = SHRTIMER_CHANNEL_BUNCH_ENTRY_REGULAR;

outcfg_para.fault_state = SHRTIMER_CHANNEL_FAULTSTATE_NONE;

outcfg_para.idle_bunch = SHRTIMER_CHANNEL_BUNCH_IDLE_DISABLE;

outcfg_para.idle_state = SHRTIMER_CHANNEL_IDLESTATE_INACTIVE;

outcfg_para.polarity = SHRTIMER_CHANNEL_POLARITY_HIGH;

outcfg_para.reset_request = SHRTIMER_CHANNEL_RESET_CMP1;

outcfg_para.set_request = SHRTIMER_CHANNEL_SET_CMP0;

shrtimer_slavetimer_waveform_channel_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
SHRTIMER_ST0_CH0, &outcfg_para);
  
```

shrtimer_slavetimer_waveform_channel_software_request

The description of shrtimer_slavetimer_waveform_channel_software_request is shown as below:

Table 3-515. Function shrtimer_slavetimer_waveform_channel_software_request

Function name	shrtimer_slavetimer_waveform_channel_software_request
Function prototype	void shrtimer_slavetimer_waveform_channel_software_request(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t channel, uint32_t request)
Function descriptions	software generates channel "set request" or "reset request"
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral

SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER timer index
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
channel	SHRTIMER timer channel index
SHRTIMER_STx_CHy(x=0..4,y=0,1)	SHRTIMER timer channel selection
Input parameter{in}	
request	request type selection
SHRTIMER_CHANNEL_SOFTWARE_SET	software event cannot generate request
SHRTIMER_CHANNEL_SOFTWARE_RESET	software event can generate request
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* software generates channel "set request" or "reset request" */

shrtimer_slavetimer_waveform_channel_software_request (SHRTIMER0,
SHRTIMER_SLAVE_TIMER0, SHRTIMER_ST0_CH0,
SHRTIMER_CHANNEL_SOFTWARE_SET);
```

shrtimer_slavetimer_waveform_channel_output_level_get

The description of shrtimer_slavetimer_waveform_channel_output_level_get is shown as below:

Table 3-516. Function shrtimer_slavetimer_waveform_channel_output_level_get

Function name	shrtimer_slavetimer_waveform_channel_output_level_get
Function prototype	uint32_t shrtimer_slavetimer_waveform_channel_output_level_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t channel);
Function descriptions	get Slave_TIMER channel output level
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection

Input parameter{in}	
timer_id	SHRTIMER timer index
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
channel	SHRTIMER timer channel index
SHRTIMER_STx_CHy(x=0..4,y=0,1)	SHRTIMER timer channel selection
Output parameter{out}	
-	-
Return value	
uint32_t	channel output level

Example:

```
/* get Slave_TIMER channel output level */

uint32_t output_level;

output_level = shrtimer_slavetimer_waveform_channel_output_level_get(uint32_t
shrtimer_periph, uint32_t timer_id, uint32_t channel)
```

shrtimer_slavetimer_waveform_channel_state_get

The description of shrtimer_slavetimer_waveform_channel_state_get is shown as below:

Table 3-517. Function shrtimer_slavetimer_waveform_channel_state_get

Function name	shrtimer_slavetimer_waveform_channel_state_get
Function prototype	uint32_t shrtimer_slavetimer_waveform_channel_state_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t channel)
Function descriptions	get Slave_TIMER channel run state
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER timer index
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
channel	SHRTIMER timer channel index

SHRTIMER_STx_CHy(x=0..4,y=0,1)	SHRTIMER timer channel selection
Output parameter{out}	
-	-
Return value	
uint32_t	SHRTIMER_CHANNEL_STATE_IDLE or SHRTIMER_CHANNEL_STATE_RUN or SHRTIMER_CHANNEL_STATE_FAULT

Example:

```
/* get Slave_TIMER channel run state */

uint32_t output_state;

output_state = shrtimer_slavetimer_waveform_channel_state_get (SHRTIMER0,
SHRTIMER_SLAVE_TIMER0, SHRTIMER_ST0_CH0);
```

shrtimer_deadtimercfg_struct_para_init

The description of shrtimer_deadtimercfg_struct_para_init is shown as below:

Table 3-518. Function shrtimer_channel_outputcfg_struct_para_init

Function name	shrtimer_deadtimercfg_struct_para_init
Function prototype	void shrtimer_deadtimercfg_struct_para_init(shrtimer_deadtimecfg_parameter_struct * dtcfg);
Function descriptions	initialize dead time configuration struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
dtcfg	shrtimer_deadtimecfg_parameter_struct, the structure members can refer to Table 3-489. Structure shrtimer_deadtimecfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize dead time configuration struct with a default value */

shrtimer_deadtimecfg_parameter_struct deadtimecfg_para;
shrtimer_deadtimercfg_struct_para_init(&deadtimecfg_para);
```

shrtimer_slavetimer_deadtime_config

The description of shrtimer_slavetimer_deadtime_config is shown as below:

Table 3-519. Function shrtimer_slavetimer_waveform_channel_software_request

Function name	shrtimer_slavetimer_deadtime_config
Function prototype	void shrtimer_slavetimer_deadtime_config(uint32_t shrtimer_periph, uint32_t timer_id, shrtimer_deadtimecfg_parameter_struct* dtcfg)
Function descriptions	configure the dead time for Slave_TIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER timer index
SHRTIMER_SLAVE_TIMERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
dtcfg	shrtimer_deadtimecfg_parameter_struct, the structure members can refer to Table 3-489. Structure shrtimer_deadtimecfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure the dead time for Slave_TIMER */

shrtimer_deadtimecfg_parameter_struct deadtimecfg_para;

shrtimer_deadtimecfg_struct_para_init(&deadtimecfg_para);

deadtimecfg_para.fallingsign_protect = SHRTIMER_DEADTIME_FALLINGSIGN_PROTECT_DISABLE;

deadtimecfg_para.falling_protect = SHRTIMER_DEADTIME_FALLING_PROTECT_DISABLE;

deadtimecfg_para.falling_sign = SHRTIMER_DEADTIME_FALLINGSIGN_POSITIVE;

deadtimecfg_para.falling_value = 0x0040;

deadtimecfg_para.prescaler = SHRTIMER_DEADTIME_PRESCALER_MUL8;

deadtimecfg_para.risingsign_protect = SHRTIMER_DEADTIME_RISINGSIGN_PROTECT_DISABLE;

deadtimecfg_para.rising_protect = SHRTIMER_DEADTIME_RISING_PROTECT_DISABLE;

deadtimecfg_para.rising_sign = SHRTIMER_DEADTIME_RISINGSIGN_POSITIVE;

deadtimecfg_para.rising_value = 0x0040;

```

```
shrtimer_slavetimer_deadtime_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
&deadtimecfg_para);
```

shrtimer_carriersignalcfg_struct_para_init

The description of shrtimer_carriersignalcfg_struct_para_init is shown as below:

Table 3-520. Function shrtimer_channel_outputcfg_struct_para_init

Function name	shrtimer_carriersignalcfg_struct_para_init
Function prototype	void shrtimer_carriersignalcfg_struct_para_init(shrtimer_carriersignalcfg_parameter_struct* carriercfg);
Function descriptions	initialize carrier signal configuration struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
carriercfg	shrtimer_carriersignalcfg_parameter_struct, the structure members can refer to Table 3-490. Structure shrtimer_carriersignalcfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize carrier signal configuration struct with a default value */

shrtimer_carriersignalcfg_parameter_struct carriercfg;
shrtimer_carriersignalcfg_struct_para_init(&carriercfg);
```

shrtimer_slavetimer_carriersignal_config

The description of shrtimer_slavetimer_carriersignal_config is shown as below:

Table 3-521. Function shrtimer_slavetimer_carriersignal_config

Function name	shrtimer_slavetimer_carriersignal_config
Function prototype	void shrtimer_slavetimer_carriersignal_config(uint32_t shrtimer_periph, uint32_t timer_id, shrtimer_carriersignalcfg_parameter_struct* carriercfg);
Function descriptions	configure the carrier signal mode for Slave_TIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	

timer_id	SHRTIMER timer index
SHRTIMER_SLAVE_TI MERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
carriercfg	shrtimer_carriersignalcfg_parameter_struct, the structure members can refer to Table 3-490. Structure shrtimer_carriersignalcfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the carrier signal mode for Slave_TIMER */
shrtimer_carriersignalcfg_parameter_struct carriercfg;
shrtimer_carriersignalcfg_struct_para_init(&carriercfg);
shrtimer_slavetimer_carriersignal_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
&carriercfg);
```

shrtimer_output_channel_enable

The description of shrtimer_output_channel_enable is shown as below:

Table 3-522. Function shrtimer_output_channel_enable

Function name	shrtimer_output_channel_enable
Function prototype	void shrtimer_output_channel_enable(uint32_t shrtimer_periph, uint32_t chid);
Function descriptions	enable a output channel
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
ch_id	SHRTIMER timer channel index
SHRTIMER_STx_CHy(x=0..4;y=0,1)	timer channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable a output channel */

void shrtimer_output_channel_enable(SHRTIMER0, SHRTIMER_ST0_CH0);

```

shrtimer_output_channel_disable

The description of shrtimer_output_channel_disable is shown as below:

Table 3-523. Function shrtimer_output_channel_disable

Function name	shrtimer_output_channel_disable
Function prototype	void shrtimer_output_channel_disable(uint32_t shrtimer_periph, uint32_t chid);
Function descriptions	disable a output channel
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
ch_id	SHRTIMER timer channel index
SHRTIMER_STx_CHy(x=0..4;y=0,1)	timer channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* disable a output channel */

void shrtimer_output_channel_disable(SHRTIMER0, SHRTIMER_ST0_CH0);

```

shrtimer_slavetimer_compare_value_config

The description of shrtimer_slavetimer_compare_value_config is shown as below:

Table 3-524. Function shrtimer_slavetimer_waveform_compare_config

Function name	shrtimer_slavetimer_compare_value_config
Function prototype	void shrtimer_slavetimer_compare_value_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t comparex, uint32_t cmpvalue);
Function descriptions	configure the compare value in Slave_TIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection

Input parameter{in}	
timer_id	SHRTIMER peripheral
SHRTIMER_SLAVE_TI MERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
comparex	compare unit selection
SHRTIMER_COMPAR Ey (y=0..4)	the compare unit y(y=0..4)
Input parameter{in}	
cmpvalue	the compare value from 3 tSHRTIMER_CK clock to 0xFFFF - (1 tSHRTIMER_CK)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the compare value in Slave_TIMER */
shrtimer_slavetimer_compare_value_config(SHRTIMERO, SHRTIMER_SLAVE_TIMER0,
SHRTIMER_COMPARE0, 3);
```

shrtimer_slavetimer_compare_value_get

The description of shrtimer_slavetimer_compare_value_get is shown as below:

Table 3-525. Function shrtimer_slavetimer_compare_value_get

Function name	shrtimer_slavetimer_compare_value_get
Function prototype	uint32_t shrtimer_slavetimer_compare_value_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t comparex)
Function descriptions	get the compare value in Slave_TIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER peripheral
SHRTIMER_SLAVE_TI MERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
comparex	compare unit selection
SHRTIMER_COMPAR Ey (y=0..4)	the compare unit y(y=0..4)

Output parameter{out}	
-	-
Return value	
uint32_t	the compare value

Example:

```
/* get the compare value in Slave_TIMER */

uint32_t cmpvalue;

cmpvalue = shrtimer_slavetimer_compare_value_get (SHRTIMER0,
SHRTIMER_SLAVE_TIMER0, SHRTIMER_COMPARE0);
```

shrtimer_mastertimer_compare_value_config

The description of shrtimer_mastertimer_compare_value_config is shown as below:

Table 3-526. Function shrtimer_mastertimer_compare_value_config

Function name	shrtimer_mastertimer_compare_value_config
Function prototype	void shrtimer_mastertimer_compare_value_config(uint32_t shrtimer_periph, uint32_t comparex, uint32_t cmpvalue);
Function descriptions	configure the compare value in Master_TIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
comparex	compare unit selection
SHRTIMER_COMPAREy (y=0..4)	the compare unit y(y=0..4)
Input parameter{in}	
cmpvalue	the compare value from 3 tSHRTIMER_CK clock to 0xFFFF - (1 tSHRTIMER_CK)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the compare value in Master_TIMER */

shrtimer_mastertimer_compare_value_config(SHRTIMER0, SHRTIMER_COMPARE0, 3);
```

shrtimer_mastertimer_compare_value_get

The description of shrtimer_mastertimer_compare_value_get is shown as below:

Table 3-527. Function shrtimer_slavetimer_compare_value_get

Function name	shrtimer_mastertimer_compare_value_get
Function prototype	uint32_t shrtimer_mastertimer_compare_value_get(uint32_t shrtimer_periph, uint32_t comparex);
Function descriptions	get the compare value in Master_TIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
comparex	compare unit selection
SHRTIMER_COMPAR _{Ey} (y=0..4)	the compare unit y(y=0..4)
Output parameter{out}	
-	-
Return value	
uint32_t	the compare value

Example:

```
/* get the compare value in Master_TIMER */

uint32_t cmpvalue;

cmpvalue = shrtimer_mastertimer_compare_value_get(SHRTIMER0,
SHRTIMER_COMPARE0)
```

shrtimer_timers_counter_value_config

The description of shrtimer_timers_counter_value_config is shown as below:

Table 3-528. Function shrtimer_timers_counter_value_config

Function name	shrtimer_timers_counter_value_config
Function prototype	void shrtimer_timers_counter_value_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t cntvalue);
Function descriptions	configure the counter value in Master_TIMER and Slave_TIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection

Input parameter{in}	
timer_id	SHRTIMER peripheral
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
cntvalue	the value ranges from 0 to 0xffff
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the counter value in Master_TIMER and Slave_TIMER */
shrtimer_timers_counter_value_config(SHRTIMER0, SHRTIMER_MASTER_TIMER, 100);
```

shrtimer_timers_counter_value_get

The description of shrtimer_timers_counter_value_get is shown as below:

Table 3-529. Function shrtimer_timers_counter_value_get

Function name	shrtimer_timers_counter_value_get
Function prototype	uint32_t shrtimer_timers_counter_value_get(uint32_t shrtimer_periph, uint32_t timer_id);
Function descriptions	get the counter value in Master_TIMER and Slave_TIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER peripheral
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Output parameter{out}	
-	-
Return value	
uint32_t	the counter value

Example:

```

/* get the counter value in Master_TIMER and Slave_TIMER */

uint32_t value;

value = shrtimer_timers_counter_value_get(SHRTIMER0, SHRTIMER_MASTER_TIMER);
  
```

shrtimer_timers_autoreload_value_config

The description of shrtimer_timers_autoreload_value_config is shown as below:

Table 3-530. Function shrtimer_timers_autoreload_value_config

Function name	shrtimer_timers_autoreload_value_config
Function prototype	void shrtimer_timers_autoreload_value_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t carvalue);
Function descriptions	configure the counter auto reload value in Master_TIMER and Slave_TIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER peripheral
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
carvalue	the value ranges from 0 to 0xffff
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure the counter auto reload value in Master_TIMER and Slave_TIMER */

shrtimer_timers_autoreload_value_config(SHRTIMER0,      SHRTIMER_MASTER_TIMER,
100);
  
```

shrtimer_timers_autoreload_value_get

The description of shrtimer_timers_autoreload_value_get is shown as below:

Table 3-531. Function shrtimer_timers_autoreload_value_get

Function name	shrtimer_timers_autoreload_value_get

Function prototype	uint32_t shrtimer_timers_autoreload_value_get(uint32_t shrtimer_periph, uint32_t timer_id)
Function descriptions	get the counter auto reload value in Master_TIMER and Slave_TIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER peripheral
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TIMERx(x=0..4)	the counter of Slave_TIMERx(x=0..4)
Output parameter{out}	
-	-
Return value	
uint32_t	the counter value

Example:

```
/* get the counter auto reload value in Master_TIMER and Slave_TIMER */

uint32_t value;

value = shrtimer_timers_autoreload_value_get(SHRTIMER0,
SHRTIMER_MASTER_TIMER);
```

shrtimer_timers_repetition_value_config

The description of shrtimer_timers_repetition_value_config is shown as below:

Table 3-532. Function shrtimer_timers_repetition_value_config

Function name	shrtimer_timers_repetition_value_config
Function prototype	void shrtimer_timers_repetition_value_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t replvalue);
Function descriptions	configure the counter repetition value in Master_TIMER and Slave_TIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER peripheral
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER

SHRTIMER_SLAVE_TI <i>MERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
Input parameter{in}	
replvalue	the value ranges from 0 to 0xffff
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the counter repetition value in Master_TIMER and Slave_TIMER */
shrtimer_timers_repetition_value_config(SHRTIMER0, SHRTIMER_MASTER_TIMER, 100);
```

shrtimer_timers_repetition_value_get

The description of shrtimer_timers_repetition_value_get is shown as below:

Table 3-533. Function shrtimer_timers_repetition_value_get

Function name	shrtimer_timers_repetition_value_get
Function prototype	uint32_t shrtimer_timers_repetition_value_get(uint32_t shrtimer_periph, uint32_t timer_id);
Function descriptions	get the counter repetition value in Master_TIMER and Slave_TIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER peripheral
SHRTIMER_MASTER_TIMER	the counter of Master_TIMER
SHRTIMER_SLAVE_TI <i>MERx(x=0..4)</i>	the counter of Slave_TIMERx(x=0..4)
Output parameter{out}	
-	-
Return value	
uint32_t	the counter value

Example:

```
/* get the counter repetition value in Master_TIMER and Slave_TIMER */
uint32_t value;
value = shrtimer_timers_repetition_value_get (SHRTIMER0, SHRTIMER_MASTER_TIMER);
```

shrtimer_exevfilter_struct_para_init

The description of shrtimer_exevfilter_struct_para_init is shown as below:

Table 3-534. Function shrtimer_exevfilter_struct_para_init

Function name	shrtimer_exevfilter_struct_para_init
Function prototype	void shrtimer_exevfilter_struct_para_init(shrtimer_exevfilter_parameter_struct * exevfilter);
Function descriptions	initialize external event filtering for Slave_TIMER configuration struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
exevfilter	shrtimer_exevfilter_parameter_struct * exevfilter, the structure members can refer to Table 3-488. Structure shrtimer_exevfilter_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize external event filtering for Slave_TIMER configuration struct with a default value */

shrtimer_exevfilter_parameter_struct exevfilter;

shrtimer_exevfilter_struct_para_init(&exevfilter);
```

shrtimer_slavetimer_exevent_filtering_config

The description of shrtimer_slavetimer_exevent_filtering_config is shown as below:

Table 3-535. Function shrtimer_slavetimer_exevent_filtering_config

Function name	shrtimer_slavetimer_exevent_filtering_config
Function prototype	void shrtimer_slavetimer_exevent_filtering_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t event_id, shrtimer_exevfilter_parameter_struct *exevfilter);
Function descriptions	configure the external event filtering for Slave_TIMER (blanking, windowing)
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	SHRTIMER timer index
SHRTIMER_SLAVE_TI	the counter of Slave_TIMERx(x=0..4)

MERx(x=0..4)	
Input parameter{in}	
event_id	SHRTIMER event index
<i>SHRTIMER_EXEVENT_NONE</i>	the counter of Slave_TIMERx <i>undefined</i> event channel
<i>SHRTIMER_EXEVENT_y</i> (y=0..9)	extern event y(y=0..9)
Input parameter{in}	
exevfilter	shrtimer_exevfilter_parameter_struct * exevfilter, the structure members can refer to Table 3-488. Structure shrtimer_exevfilter parameter struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure the external event filtering for Slave_TIMER (blanking, windowing) */

shrtimer_exevfilter_parameter_struct exevfilter;
shrtimer_exevfilter_struct_para_init(&exevfilter);
exevfilter.filter_mode = SHRTIMER_EXEVFILTER_BLANKINGCMP1;
exevfilter.memorized = SHRTIMER_EXEVMEMORIZED_DISABLE;
shrtimer_slavetimer_exevent_filtering_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
SHRTIMER_EXEVENT_5, &exevfilter);

```

shrtimer_exeventcfg_struct_para_init

The description of shrtimer_exeventcfg_struct_para_init is shown as below:

Table 3-536. Function shrtimer_exeventcfg_struct_para_init

Function name	shrtimer_exeventcfg_struct_para_init
Function prototype	void shrtimer_exeventcfg_struct_para_init(shrtimer_exeventcfg_parameter_struct t * exevcfg);
Function descriptions	initialize external event configuration struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
exevcfg	shrtimer_exeventcfg_parameter_struct, the structure members can refer to Table 3-493. Structure shrtimer_exeventcfg_parameter_struct
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* initialize external event configuration struct with a default value */

shrtimer_exeventcfg_parameter_struct exevcfg;
shrtimer_exeventcfg_struct_para_init(&exevcfg);
```

shrtimer_exevent_config

The description of shrtimer_exevent_config is shown as below:

Table 3-537. Function shrtimer_exevent_config

Function name	shrtimer_exevent_config
Function prototype	void shrtimer_exevent_config(uint32_t shrtimer_periph, uint32_t event_id, shrtimer_exeventcfg_parameter_struct* exevcfg);
Function descriptions	configure the an external event
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
event_id	SHRTIMER event index
SHRTIMER_EXEVENT_NONE	the counter of Slave_TIMERx <i>undefined event channel</i>
SHRTIMER_EXEVENT_y(y=0..9)	extern event y(y=0..9)
Input parameter{in}	
exevcfg	shrtimer_exeventcfg_parameter_struct, the structure members can refer to Table 3-493. Structure shrtimer_exeventcfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the an external event */

shrtimer_exeventcfg_parameter_struct exevcfg;
shrtimer_exeventcfg_struct_para_init(&exevcfg);
exevcfg.digital_filter = 0x5;
```

```

exevcfg.edge = SHRTIMER_EXEV_EDGE_RISING;
exevcfg.polarity = SHRTIMER_EXEV_EDGE_LEVEL;
exevcfg.source = SHRTIMER_EXEV_SRC0;
shrtimer_exevent_config(SHRTIMER0, SHRTIMER_EXEVENT_5, &exevcfg);
  
```

shrtimer_exevent_prescaler

The description of shrtimer_exevent_prescaler is shown as below:

Table 3-538. Function shrtimer_exevent_prescaler

Function name	shrtimer_exevent_prescaler
Function prototype	void shrtimer_exevent_prescaler(uint32_t shrtimer_periph, uint32_t prescaler);
Function descriptions	configure external event digital filter clock division
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
prescaler	clock division value
SHRTIMER_EXEV_PR ESCALER_DIVx(x=1,2, 4,8)	fSHRTIMER_EXEVFCK = fSHRTIMER_CK/x (x=1,2,4,8)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure external event digital filter clock division */

shrtimer_exevent_prescaler(SHRTIMER0, SHRTIMER_EXEV_PRESCALER_DIV1);
  
```

shrtimer_synccfg_struct_para_init

The description of shrtimer_synccfg_struct_para_init is shown as below:

Table 3-539. Function shrtimer_synccfg_struct_para_init

Function name	shrtimer_synccfg_struct_para_init
Function prototype	void shrtimer_synccfg_struct_para_init(shrtimer_synccfg_parameter_struct* synccfg);
Function descriptions	initialize synchronization configuration struct with a default value
Precondition	-

The called functions		-
Input parameter{in}		
syncfg	shrtimer_syncfg_parameter_struct, the structure members can refer to Table 3-491. Structure shrtimer_syncfg_parameter_struct	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

```
/* initialize synchronization configuration struct with a default value */
shrtimer_syncfg_parameter_struct syncfg;
shrtimer_syncfg_struct_para_init (&syncfg);
```

shrtimer_synchronization_config

The description of shrtimer_synchronization_config is shown as below:

Table 3-540. Function shrtimer_synchronization_config

Function name	shrtimer_synchronization_config
Function prototype	void shrtimer_synchronization_config(uint32_t shrtimer_periph, shrtimer_syncfg_parameter_struct* syncfg);
Function descriptions	configure the synchronization input/output of the SHRTIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
syncfg	shrtimer_syncfg_parameter_struct, the structure members can refer to Table 3-491. Structure shrtimer_syncfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the synchronization input/output of the SHRTIMER */
shrtimer_syncfg_parameter_struct syncfg;
shrtimer_syncfg_struct_para_init (&syncfg);
syncfg.input_source = SHRTIMER_SYNCINPUTSOURCE_EXTERNAL;
```

```
shrtimer_synchronization_config(SHRTIMER0, &syncfg);
```

shrtimer_faultcfg_struct_para_init

The description of shrtimer_faultcfg_struct_para_init is shown as below:

Table 3-541. Function shrtimer_faultcfg_struct_para_init

Function name	shrtimer_faultcfg_struct_para_init
Function prototype	void shrtimer_faultcfg_struct_para_init(shrtimer_faultcfg_parameter_struct *faultcfg);
Function descriptions	initialize fault input configuration struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
faultcfg	shrtimer_faultcfg_parameter_struct, the structure members can refer to Table 3-494. Structure shrtimer_faultcfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize fault input configuration struct with a default value */

shrtimer_faultcfg_parameter_struct faultcfg;
shrtimer_syncfg_struct_para_init (&faultcfg);
```

shrtimer_fault_config

The description of shrtimer_fault_config is shown as below:

Table 3-542. Function shrtimer_fault_config

Function name	shrtimer_fault_config
Function prototype	void shrtimer_fault_config(uint32_t shrtimer_periph, uint32_t fault_id, shrtimer_faultcfg_parameter_struct* faultcfg);
Function descriptions	configure the fault input
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
fault_id	SHRTIMER fault index
SHRTIMER_FAULT_y(y=0..4)	SHRTIMER fault selection

Input parameter{in}	
faultcfg	shrtimer_faultcfg_parameter_struct, the structure members can refer to Table 3-494. Structure shrtimer_faultcfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the fault input */

shrtimer_faultcfg_parameter_struct faultcfg;

shrtimer_syncfg_struct_para_init (&faultcfg);

faultcfg_para.control = SHRTIMER_FAULT_CHANNEL_ENABLE;

faultcfg_para.filter = 0x0;

faultcfg_para.polarity = SHRTIMER_FAULT_POLARITY_HIGH;

faultcfg_para.protect = SHRTIMER_FAULT_PROTECT_ENABLE;

faultcfg_para.source = SHRTIMER_FAULT_SOURCE_PIN;

shrtimer_fault_config(SHRTIMER0, SHRTIMER_FAULT_0, &faultcfg);
```

shrtimer_fault_prescaler_config

The description of shrtimer_fault_prescaler_config is shown as below:

Table 3-543. Function shrtimer_fault_prescaler_config

Function name	shrtimer_fault_prescaler_config
Function prototype	void shrtimer_fault_prescaler_config(uint32_t shrtimer_periph, uint32_t prescaler);
Function descriptions	configure the fault input digital filter clock division
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
prescaler	clock division value
SHRTIMER_FAULT_P RESCALER_DIVy(y=1, 2,4,8)	fSHRTIMER_FLTFCK = fSHRTIMER_CK/y (y=1,2,4,8)
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure the fault input digital filter clock division */

void shrtimer_fault_prescaler_config(SHRTIMER0,
SHRTIMER_FAULT_PRESCALER_DIV1);
```

shrtimer_fault_input_enable

The description of shrtimer_fault_input_enable is shown as below:

Table 3-544. Function shrtimer_fault_input_enable

Function name	shrtimer_fault_input_enable
Function prototype	void shrtimer_fault_input_enable(uint32_t shrtimer_periph, uint32_t fault_id);
Function descriptions	fault input enable
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
fault_id	SHRTIMER fault index
SHRTIMER_FAULT_y(<i>y=0..4</i>)	SHRTIMER fault selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* fault input enable */

shrtimer_fault_input_enable(SHRTIMER0, SHRTIMER_FAULT_0);
```

shrtimer_fault_input_disable

The description of shrtimer_fault_input_disable is shown as below:

Table 3-545. Function shrtimer_fault_input_disable

Function name	shrtimer_fault_input_disable
Function prototype	void shrtimer_fault_input_disable(uint32_t shrtimer_periph, uint32_t fault_id);
Function descriptions	fault input disable

Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
fault_id	SHRTIMER fault index
SHRTIMER_FAULT_y(y=0..4)	SHRTIMER fault selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* fault input disable */
shrtimer_fault_input_disable(SHRTIMER0, SHRTIMER_FAULT_0);
```

shrtimer_timers_dma_enable

The description of shrtimer_timers_dma_enable is shown as below:

Table 3-546. Function shrtimer_timers_dma_enable

Function name	shrtimer_timers_dma_enable
Function prototype	void shrtimer_timers_dma_enable(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t dmareq);
Function descriptions	enable the Master_TIMER and Slave_TIMER DMA request
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	master timer and slave timer index
SHRTIMER_MASTER_TIMER	the master timer
SHRTIMER_SLAVE_TIMERx	the slave timer selection(x=0..4)
Input parameter{in}	
dmareq	DMA request source
SHRTIMER_MT_ST_DMA_CMPy(y=0..3)	compare y DMA request, for Master_TIMER and Slave_TIMER (y=0..3)
SHRTIMER_MT_ST_D	repetition DMA request, for Master_TIMER and Slave_TIMER

MA_REP	
SHRTIMER_MT_DMA_SYNID	synchronization input DMA request, for Master_TIMER
SHRTIMER_MT_ST_DMMA_UPD	update DMA request, for Master_TIMER and Slave_TIMER
SHRTIMER_ST_DMA_CAPy(y=0,1)	capture y DMA request, for Slave_TIMER(y=0,1)
SHRTIMER_ST_DMA_CHyOA(y=0,1)	channel y output active DMA request, for Slave_TIMER(y=0,1)
SHRTIMER_ST_DMA_CHyONA(y=0,1)	channel y output inactive DMA request, for Slave_TIMER(y=0,1)
SHRTIMER_ST_DMA_CNTRST	counter reset DMA request, for Slave_TIMER
SHRTIMER_ST_DMA_DLYIDLE	delayed IDLE mode entry DMA request, for Slave_TIMER
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the Master_TIMER and Slave_TIMER DMA request */
shrtimer_timers_dma_enable(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_DMA_CMP0);
```

shrtimer_timers_dma_disable

The description of shrtimer_timers_dma_disable is shown as below:

Table 3-547. Function shrtimer_timers_dma_enable

Function name	shrtimer_timers_dma_disable
Function prototype	void shrtimer_timers_dma_disable(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t dmareq);
Function descriptions	disable the Master_TIMER and Slave_TIMER DMA request
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	master timer and slave timer index
SHRTIMER_MASTER_TIMER	the master timer

<i>SHRTIMER_SLAVE_TI</i> <i>MERx</i>	the slave timer selection(x=0..4)
Input parameter{in}	
<i>dmareq</i>	DMA request source
<i>SHRTIMER_MT_ST_D</i> <i>MA_CMPy(y=0..3)</i>	compare y DMA request, for Master_TIMER and Slave_TIMER (y=0..3)
<i>SHRTIMER_MT_ST_D</i> <i>MA_REP</i>	repetition DMA request, for Master_TIMER and Slave_TIMER
<i>SHRTIMER_MT_DMA_</i> <i>SYNID</i>	synchronization input DMA request, for Master_TIMER
<i>SHRTIMER_MT_ST_D</i> <i>MA_UPD</i>	update DMA request, for Master_TIMER and Slave_TIMER
<i>SHRTIMER_ST_DMA_</i> <i>CAPy(y=0,1)</i>	capture y DMA request, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_DMA_</i> <i>CHyOA(y=0,1)</i>	channel y output active DMA request, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_DMA_</i> <i>CHyONA(y=0,1)</i>	channel y output inactive DMA request, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_DMA_</i> <i>CNTRST</i>	counter reset DMA request, for Slave_TIMER
<i>SHRTIMER_ST_DMA_</i> <i>DLYIDLE</i>	delayed IDLE mode entry DMA request, for Slave_TIMER
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the Master_TIMER and Slave_TIMER DMA request */

shrtimer_timers_dma_disable(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_DMA_CMP0);
```

shrtimer_dmamode_config

The description of shrtimer_dmamode_config is shown as below:

Table 3-548. Function shrtimer_dmamode_config

Function name	shrtimer_dmamode_config
Function prototype	void shrtimer_dmamode_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t regupdate);
Function descriptions	configure the DMA mode for Master_TIMER or Slave_TIMER
Precondition	-
The called functions	-

Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	master timer and slave timer index
SHRTIMER_MASTER_TIMER	the master timer
SHRTIMER_SLAVE_TIMERx	the slave timer selection(x=0..4)
Input parameter{in}	
regupdate	registers to be updated
SHRTIMER_DMAMOD_E_NONE	No register is updated by DMA mode, for Master_TIMER and Slave_TIMER
SHRTIMER_DMAMOD_E_CTL0	MTCTL0 or STxCTL0 register is updated by DMA mode, for Master_TIMER and Slave_TIMER
SHRTIMER_DMAMOD_E_INTC	MT or STx register is updated by DMA mode, for Master_TIMER and Slave_TIMER
SHRTIMER_DMAMOD_E_DMAINTEN	MTINTC or STxINTC register is updated by DMA mode, for Master_TIMER and Slave_TIMER
SHRTIMER_DMAMOD_E_CNT	MTCNT or STxCNT register is updated by DMA mode, for Master_TIMER and Slave_TIMER
SHRTIMER_DMAMOD_E_CAR	MTCAR or STxCAR register is updated by DMA mode, for Master_TIMER and Slave_TIMER
SHRTIMER_DMAMOD_E_CREP	MTCREP or STxCREP register is updated by DMA mode, for Master_TIMER and Slave_TIMER
SHRTIMER_DMAMOD_E_CMPyV(y=0..3)	MTCMPyV or STxCMPyV register is updated by DMA mode, for Master_TIMER and Slave_TIMER(y=0..3)
SHRTIMER_DMAMOD_E_DTCTL	STxDTCTL register is updated by DMA mode, only for Slave_TIMER
SHRTIMER_DMAMOD_E_CHySET(y=0,1)	STxCHySET register is updated by DMA mode, only for Slave_TIMER(y=0,1)
SHRTIMER_DMAMOD_E_CHyRST(y=0,1)	STxCHyRST register is updated by DMA mode, only for Slave_TIMER(y=0,1)
SHRTIMER_DMAMOD_E_EXEVFCFGy(y=0,1)	STxEXEVFCFGy register is updated by DMA mode, only for Slave_TIMER(y=0,1)
SHRTIMER_DMAMOD_E_CNTRST	STxCNTRST register is updated by DMA mode, only for Slave_TIMER
SHRTIMER_DMAMOD_E_CSCTL	STxCSCTL register is updated by DMA mode, only for Slave_TIMER
SHRTIMER_DMAMOD_E_CHOCTL	STxCHOCTL register is updated by DMA mode, only for Slave_TIMER
SHRTIMER_DMAMOD_E_FLTCTL	STxFLTCTL register is updated by DMA mode, only for Slave_TIMER

SHRTIMER_DMAMOD_E_ACTL	STxACTL register is updated by DMA mode, only for Slave_TIMER
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the Master_TIMER and Slave_TIMER DMA request */

shrtimer_dmamode_config(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_DMAMODE_NONE);
```

shrtimer_bunchmode_struct_para_init

The description of shrtimer_bunchmode_struct_para_init is shown as below:

Table 3-549. Function shrtimer_bunchmode_struct_para_init

Function name	shrtimer_bunchmode_struct_para_init
Function prototype	void shrtimer_bunchmode_struct_para_init(shrtimer_bunchmode_parameter_struct* bmcfg);
Function descriptions	initialize bunch mode configuration struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
bmcfg	shrtimer_bunchmode_parameter_struct, the structure members can refer to Table 3-492. Structure shrtimer_bunchmode_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize bunch mode configuration struct with a default value */

shrtimer_bunchmode_parameter_struct bmcfg;
shrtimer_bunchmode_struct_para_init(&bmcfg);
```

shrtimer_bunchmode_config

The description of shrtimer_bunchmode_config is shown as below:

Table 3-550. Function shrtimer_bunchmode_config

Function name	shrtimer_bunchmode_config
----------------------	---------------------------

Function prototype	void shrtimer_bunchmode_config(uint32_t shrtimer_periph, shrtimer_bunchmode_parameter_struct* bmcfg);
Function descriptions	configure bunch mode for the SHRTIMER
Precondition	-
The called functions	-
Input parameter{in}	
bmcfg	shrtimer_bunchmode_parameter_struct, the structure members can refer to Table 3-492. Structure shrtimer_bunchmode_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure bunch mode for the SHRTIMER */

shrtimer_bunchmode_parameter_struct bmcfg;

shrtimer_bunchmode_struct_para_init(&bmcfg);

bmcfg.clock_source = SHRTIMER_BUNCHMODE_CLOCKSOURCE_ST0;

bmcfg.idle_duration = 3;

bmcfg.mode = SHRTIMER_BUNCHMODE_CONTINOUS;

bmcfg.period = 6;

bmcfg.prescaler = SHRTIMER_BUNCHMODE_PRESCALER_DIV1;

bmcfg.shadow = SHRTIMER_BUNCHMODEPRELOAD_DISABLED;

bmcfg.trigger = SHRTIMER_BUNCHMODE_TRIGGER_SOFTWARE;

shrtimer_bunchmode_config(SHRTIMER0, &bmcfg);
```

shrtimer_bunchmode_enable

The description of shrtimer_bunchmode_enable is shown as below:

Table 3-551. Function shrtimer_bunchmode_enable

Function name	shrtimer_bunchmode_enable
Function prototype	void shrtimer_bunchmode_enable(uint32_t shrtimer_periph);
Function descriptions	enable bunch mode for the SHRTIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable bunch mode for the SHRTIMER */

shrtimer_bunchmode_enable(SHRTIMER0);
```

shrtimer_bunchmode_disable

The description of shrtimer_bunchmode_disable is shown as below:

Table 3-552. Function shrtimer_bunchmode_disable

Function name	shrtimer_bunchmode_disable
Function prototype	void shrtimer_bunchmode_disable(uint32_t shrtimer_periph);
Function descriptions	disable bunch mode for the SHRTIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable bunch mode for the SHRTIMER */

shrtimer_bunchmode_disable(SHRTIMER0);
```

shrtimer_bunchmode_flag_get

The description of shrtimer_bunchmode_flag_get is shown as below:

Table 3-553. Function shrtimer_bunchmode_flag_get

Function name	shrtimer_bunchmode_flag_get
Function prototype	uint32_t shrtimer_bunchmode_flag_get(uint32_t shrtimer_periph);
Function descriptions	get bunch mode operating flag
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection

Output parameter{out}	
-	-
Return value	
uint32_t	SHRTIMER_BUNCHMODE_OPERATION_OFF or SHRTIMER_BUNCHMODE_OPERATION_ON

Example:

```
/* get bunch mode operating flag */

uint32_t flag;

flag = shrtimer_bunchmode_flag_get(SHRTIMER0);
```

shrtimer_bunchmode_software_start

The description of shrtimer_bunchmode_software_start is shown as below:

Table 3-554. Function shrtimer_bunchmode_software_start

Function name	shrtimer_bunchmode_software_start
Function prototype	void shrtimer_bunchmode_software_start(uint32_t shrtimer_periph);
Function descriptions	bunch mode started by software
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* bunch mode started by software */

shrtimer_bunchmode_software_start(SHRTIMER0);
```

shrtimer_slavetimer_capture_config

The description of shrtimer_slavetimer_capture_config is shown as below:

Table 3-555. Function shrtimer_slavetimer_capture_config

Function name	shrtimer_slavetimer_capture_config
Function prototype	void shrtimer_slavetimer_capture_config(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t capturex, uint32_t trgsource);
Function descriptions	configure the capture source in Slave_TIMER
Precondition	-

The called functions		-
Input parameter{in}		
shrtimer_periph	SHRTIMER peripheral	
SHRTIMERx(x=0)	SHRTIMER selection	
Input parameter{in}		
timer_id	master timer and slave timer index	
SHRTIMER_SLAVE_TI MERx	the slave timer selection(x=0..4)	
Input parameter{in}		
capturex	capture unit index	
SHRTIMER_CAPTURE _y(y=0,1)	capture unit selection(y=0,1)	
Input parameter{in}		
trgsource	capture trigger source	
SHRTIMER_CAPTURE TRIGGER_NONE	Capture trigger is disabled	
SHRTIMER_CAPTURE TRIGGER_UPDATE	capture triggered by update event	
SHRTIMER_CAPTURE TRIGGER_EXEV_y(y=0..9)	capture triggered by external event 0(y=0..9)	
SHRTIMER_CAPTURE TRIGGER_STy_ACTIV E(y=0..4)	capture triggered by STyCH0_O output inactive to active transition(y=0..4)	
SHRTIMER_CAPTURE TRIGGER_STy_INACTI VE(y=0..4)	capture triggered by STyCH0_O output active to inactive transition(y=0..4)	
SHRTIMER_CAPTURE TRIGGER_STy_CMP0(y=0..4)	capture triggered by compare 0 event of Slave_TIMERy(y=0..4)	
SHRTIMER_CAPTURE TRIGGER_STy_CMP1(y=0..4)	capture triggered by compare 1 event of Slave_TIMERy(y=0..4)	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

```
/* configure the capture source in Slave_TIMER */
shrtimer_slavetimer_capture_config(SHRTIMER0, SHRTIMER_SLAVE_TIMER0,
SHRTIMER_CAPTURE_0, SHRTIMER_CAPTURETRIGGER_NONE);
```

shrtimer_slavetimer_capture_software

The description of shrtimer_slavetimer_capture_software is shown as below:

Table 3-556. Function shrtimer_slavetimer_capture_software

Function name	shrtimer_slavetimer_capture_software
Function prototype	void shrtimer_slavetimer_capture_software(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t capturex);
Function descriptions	configure the capture source in Slave_TIMER
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	master timer and slave timer index
SHRTIMER_SLAVE_TI MERx	the slave timer selection(x=0..4)
Input parameter{in}	
capturex	capture unit index
SHRTIMER_CAPTURE _y(y=0,1)	capture unit selection(y=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the capture source in Slave_TIMER */

shrtimer_slavetimer_capture_software(SHRTIMER0, SHRTIMER_SLAVE
SHRTIMER_CAPTURE_0);
```

shrtimer_slavetimer_capture_value_read

The description of shrtimer_slavetimer_capture_value_read is shown as below:

Table 3-557. Function shrtimer_slavetimer_capture_value_read

Function name	shrtimer_slavetimer_capture_value_read
Function prototype	uint32_t shrtimer_slavetimer_capture_value_read(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t capturex);
Function descriptions	read the capture value
Precondition	-
The called functions	-
Input parameter{in}	

shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	master timer and slave timer index
SHRTIMER_SLAVE_TI MERx	the slave timer selection(x=0..4)
Input parameter{in}	
capturex	capture unit index
SHRTIMER_CAPTURE _y(y=0,1)	capture unit selection(y=0,1)
Output parameter{out}	
-	-
Return value	
uint32_t	capture value

Example:

```
/* read the capture value */

uint32_t capture_value;

capture_value      =      shrtimer_slavetimer_capture_value_read      (SHRTIMER0,
SHRTIMER_SLAVE SHRTIMER_CAPTURE_0);
```

shrtimer_adctrigcfg_struct_para_init

The description of shrtimer_adctrigcfg_struct_para_init is shown as below:

Table 3-558. Function shrtimer_adctrigcfg_struct_para_init

Function name	shrtimer_adctrigcfg_struct_para_init
Function prototype	void shrtimer_adctrigcfg_struct_para_init(shrtimer_adctrigcfg_parameter_struct* triggercfg);
Function descriptions	initialize ADC trigger configuration struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
triggercfg	shrtimer_adctrigcfg_parameter_struct, the structure members can refer to Table 3-495. Structure shrtimer_adctrigcfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize ADC trigger configuration struct with a default value */
```

```

shrtimer_adctrigcfg_parameter_struct triggercfg;
shrtimer_adctrigcfg_struct_para_init(&triggercfg);
  
```

shrtimer_adc_trigger_config

The description of shrtimer_adc_trigger_config is shown as below:

Table 3-559. Function shrtimer_adc_trigger_config

Function name	shrtimer_adc_trigger_config
Function prototype	void shrtimer_adc_trigger_config(uint32_t shrtimer_periph, uint32_t trigger_id, shrtimer_adctrigcfg_parameter_struct* triggercfg);
Function descriptions	configure the trigger source to ADC and the update source
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
trigger_id	ADC trigger event
SHRTIMER_ADCTRIG_y(y=0..3)	the slave timer selection(x=0..4)
Input parameter{in}	
triggercfg	shrtimer_adctrigcfg_parameter_struct, the structure members can refer to Table 3-495. Structure shrtimer_adctrigcfg_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure the trigger source to ADC and the update source */

shrtimer_adctrigcfg_parameter_struct triggercfg;
shrtimer_adctrigcfg_struct_para_init(&triggercfg);

triggercfg.update_source = SHRTIMER_ADCTRGI_UPDATE_MT;
triggercfg.trigger = SHRTIMER_ADCTRGI02_EVENT_NONE;
shrtimer_adc_trigger_config(SHRTIMER0, SHRTIMER_ADCTRIG_0, &triggercfg);
  
```

shrtimer_timers_flag_get

The description of shrtimer_timers_flag_get is shown as below:

Table 3-560. Function shrtimer_timers_flag_get

Function name	shrtimer_timers_flag_get
Function prototype	FlagStatus shrtimer_timers_flag_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t flag);
Function descriptions	get the Master_TIMER and Slave_TIMER flag
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	master timer and slave timer index
SHRTIMER_MASTER_TIMER	the master timer
SHRTIMER_SLAVE_TIMERx	the slave timer selection(x=0..4)
Input parameter{in}	
flag	the flag source
SHRTIMER_MT_ST_F_LAG_CMPy(y=0..3)	compare y flag, for Master_TIMER and Slave_TIMER (y=0..3)
SHRTIMER_MT_ST_F_LAG_REP	repetition flag, for Master_TIMER and Slave_TIMER
SHRTIMER_MT_INT_F_LAG_SYN	synchronization input flag, for Master_TIMER
SHRTIMER_MT_ST_F_LAG_UPD	update flag, for Master_TIMER and Slave_TIMER
SHRTIMER_ST_FLAG_CAPy(y=0,1)	capture y flag, for Slave_TIMER(y=0,1)
SHRTIMER_ST_FLAG_CHyOA(y=0,1)	channel y output active flag, for Slave_TIMER(y=0,1)
SHRTIMER_ST_FLAG_CHyONA(y=0,1)	channel y output inactive flag, for Slave_TIMER(y=0,1)
SHRTIMER_ST_FLAG_CNTRST	counter reset flag, for Slave_TIMER
SHRTIMER_ST_FLAG_DLYIDLE	delayed IDLE mode entry flag, for Slave_TIMER
SHRTIMER_ST_FLAG_CBLN	current balanced flag, for Slave_TIMER
SHRTIMER_ST_FLAG_BLNIDLE	balanced IDLE flag, for Slave_TIMER
SHRTIMER_ST_FLAG_CHyOUT(y=0,1)	channel y output flag, for Slave_TIMER(y=0,1)
Output parameter{out}	

-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the Master_TIMER and Slave_TIMER flag */

FlagStatus flag = RESET;

flag = shrtimer_timers_flag_get(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_FLAG_CMP0);
```

shrtimer_timers_flag_clear

The description of shrtimer_timers_flag_clear is shown as below:

Table 3-561. Function shrtimer_timers_flag_clear

Function name	shrtimer_timers_flag_clear
Function prototype	void shrtimer_timers_flag_clear(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t flag);
Function descriptions	clear the Master_TIMER and Slave_TIMER flag
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	master timer and slave timer index
SHRTIMER_MASTER_TIMER	the master timer
SHRTIMER_SLAVE_TIMERx	the slave timer selection(x=0..4)
Input parameter{in}	
flag	the flag source
SHRTIMER_MT_ST_FLAG_CMPy(y=0..3)	compare y flag, for Master_TIMER and Slave_TIMER (y=0..3)
SHRTIMER_MT_ST_FLAG REP	repetition flag, for Master_TIMER and Slave_TIMER
SHRTIMER_MT_INT_FLAG_SYN	synchronization input flag, for Master_TIMER
SHRTIMER_MT_ST_FLAG_UPD	update flag, for Master_TIMER and Slave_TIMER
SHRTIMER_ST_FLAG_CAPy(y=0,1)	capture y flag, for Slave_TIMER(y=0,1)
SHRTIMER_ST_FLAG	channel y output active flag, for Slave_TIMER(y=0,1)

<code>_CHyOA(y=0,1)</code>	
<code>SHRTIMER_ST_FLAG</code> <code>_CHyONA(y=0,1)</code>	channel y output inactive flag, for Slave_TIMER(y=0,1)
<code>SHRTIMER_ST_FLAG</code> <code>_CNTRST</code>	counter reset flag, for Slave_TIMER
<code>SHRTIMER_ST_FLAG</code> <code>_DLYIDLE</code>	delayed IDLE mode entry flag, for Slave_TIMER
<code>SHRTIMER_ST_FLAG</code> <code>_CBLN</code>	current balanced flag, for Slave_TIMER
<code>SHRTIMER_ST_FLAG</code> <code>_BLNIDLE</code>	balanced IDLE flag, for Slave_TIMER
<code>SHRTIMER_ST_FLAG</code> <code>_CHyOUT(y=0,1)</code>	channel y output flag, for Slave_TIMER(y=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the Master_TIMER and Slave_TIMER flag */

shrtimer_timers_flag_clear(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_FLAG_CMP0);
```

shrtimer_common_flag_get

The description of shrtimer_common_flag_get is shown as below:

Table 3-562. Function shrtimer_common_flag_get

Function name	shrtimer_common_flag_get
Function prototype	FlagStatus shrtimer_common_flag_get(uint32_t shrtimer_periph, uint32_t flag);
Function descriptions	get the common flag
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
flag	the flag source
SHRTIMER_FLAG_FLT y(y=0..3)	fault y interrupt flag (y=0..3)
SHRTIMER_FLAG_SY SFLT	system fault interrupt flag

SHRTIMER_FLAG_DL LCAL	DLL calibration completed interrupt flag
SHRTIMER_FLAG_BM PER	bunch mode period interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the common flag */

FlagStatus flag = RESET;

flag = shrtimer_common_flag_get(SHRTIMER0, SHRTIMER_FLAG_FLT0);
```

shrtimer_common_flag_clear

The description of shrtimer_common_flag_clear is shown as below:

Table 3-563. Function shrtimer_common_flag_clear

Function name	shrtimer_common_flag_clear
Function prototype	void shrtimer_common_flag_clear(uint32_t shrtimer_periph, uint32_t flag);
Function descriptions	clear the common flag
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
flag	the flag source
SHRTIMER_FLAG_FLT y(y=0..3)	fault y interrupt flag (y=0..3)
SHRTIMER_FLAG_SY SFLT	system fault interrupt flag
SHRTIMER_FLAG_DL LCAL	DLL calibration completed interrupt flag
SHRTIMER_FLAG_BM PER	bunch mode period interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* clear the common flag */

shrtimer_common_flag_clear(SHRTIMER0, SHRTIMER_FLAG_FLT0);

```

shrtimer_timers_interrupt_enable

The description of shrtimer_timers_interrupt_enable is shown as below:

Table 3-564. Function shrtimer_timers_interrupt_enable

Function name	shrtimer_timers_interrupt_enable
Function prototype	void shrtimer_timers_interrupt_enable(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t interrupt);
Function descriptions	enable the Master_TIMER and Slave_TIMER interrupt
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	master timer and slave timer index
SHRTIMER_MASTER_TIMER	the master timer
SHRTIMER_SLAVE_TIMERx	the slave timer selection(x=0..4)
Input parameter{in}	
interrupt	the interrupt source
SHRTIMER_MT_ST_IN_T_CMPy(y=0..3)	compare y interrupt, for Master_TIMER and Slave_TIMER (y=0..3)
SHRTIMER_MT_ST_IN_T_REP	repetition interrupt, for Master_TIMER and Slave_TIMER
SHRTIMER_MT_INT_SYNI	synchronization input interrupt, for Master_TIMER
SHRTIMER_MT_ST_IN_T_UPD	update interrupt, for Master_TIMER and Slave_TIMER
SHRTIMER_ST_INT_CAPy(y=0,1)	capture y interrupt, for Slave_TIMER(y=0,1)
SHRTIMER_ST_INT_CHyOA(y=0,1)	channel y output active interrupt, for Slave_TIMER(y=0,1)
SHRTIMER_ST_INT_CHyONA(y=0,1)	channel y output inactive interrupt, for Slave_TIMER(y=0,1)
SHRTIMER_ST_INT_CNT_RST	counter reset interrupt, for Slave_TIMER
SHRTIMER_ST_INT_DELAY_IDLE	delayed IDLE mode entry interrupt, for Slave_TIMER

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the Master_TIMER and Slave_TIMER interrupt */

shrtimer_timers_interrupt_enable(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_INT_CMP0);
```

shrtimer_timers_interrupt_disable

The description of shrtimer_timers_interrupt_disable is shown as below:

Table 3-565. Function shrtimer_timers_interrupt_disable

Function name	shrtimer_timers_interrupt_disable
Function prototype	void shrtimer_timers_interrupt_disable(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t interrupt);
Function descriptions	disable the Master_TIMER and Slave_TIMER interrupt
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	master timer and slave timer index
SHRTIMER_MASTER_TIMER	the master timer
SHRTIMER_SLAVE_TIMERx	the slave timer selection(x=0..4)
Input parameter{in}	
interrupt	the interrupt source
SHRTIMER_MT_ST_IN_T_CMPy(y=0..3)	compare y interrupt, for Master_TIMER and Slave_TIMER (y=0..3)
SHRTIMER_MT_ST_IN_T_REP	repetition interrupt, for Master_TIMER and Slave_TIMER
SHRTIMER_MT_INT_SYN	synchronization input interrupt, for Master_TIMER
SHRTIMER_MT_ST_IN_T_UPD	update interrupt, for Master_TIMER and Slave_TIMER
SHRTIMER_ST_INT_CAPy(y=0,1)	capture y interrupt, for Slave_TIMER(y=0,1)
SHRTIMER_ST_INT_CHANy	channel y output active interrupt, for Slave_TIMER(y=0,1)

<i>HyOA(y=0,1)</i>	
<i>SHRTIMER_ST_INT_C</i> <i>HyONA(y=0,1)</i>	channel y output inactive interrupt, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_INT_C</i> <i>NTRST</i>	counter reset interrupt, for Slave_TIMER
<i>SHRTIMER_ST_INT_D</i> <i>LYIDLE</i>	delayed IDLE mode entry interrupt, for Slave_TIMER
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the Master_TIMER and Slave_TIMER interrupt */

shrtimer_timers_interrupt_disable(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_INT_CMP0);
```

shrtimer_timers_interrupt_flag_get

The description of shrtimer_timers_interrupt_flag_get is shown as below:

Table 3-566. Function shrtimer_timers_interrupt_flag_get

Function name	shrtimer_timers_interrupt_flag_get
Function prototype	FlagStatus shrtimer_timers_interrupt_flag_get(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t interrupt);
Function descriptions	get the Master_TIMER and Slave_TIMER interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	master timer and slave timer index
SHRTIMER_MASTER_TIMER	the master timer
SHRTIMER_SLAVE_TIMERx	the slave timer selection(x=0..4)
Input parameter{in}	
interrupt	the interrupt source
SHRTIMER_MT_ST_IN <i>T</i> <i>_FLAG_CMPy(y=0..3)</i>	compare y interrupt flag, for Master_TIMER and Slave_TIMER (y=0..3)
SHRTIMER_MT_ST_IN	repetition interrupt flag, for Master_TIMER and Slave_TIMER

<i>T_FLAG REP</i>	
<i>SHRTIMER_MT_INT_F LAG_SYN</i>	synchronization input interrupt flag, for Master_TIMER
<i>SHRTIMER_MT_ST_IN T_FLAG_UPD</i>	update interrupt flag, for Master_TIMER and Slave_TIMER
<i>SHRTIMER_ST_INT_F LAG_CAPy(y=0,1)</i>	capture y interrupt flag, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_INT_F LAG_CHyOA(y=0,1)</i>	channel y output active interrupt flag, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_INT_F LAG_CHyONA(y=0,1)</i>	channel y output inactive interrupt flag, for Slave_TIMER(y=0,1)
<i>SHRTIMER_ST_INT_F LAG_CNTRST</i>	counter reset interrupt flag, for Slave_TIMER
<i>SHRTIMER_ST_INT_F LAG_DLYIDLE</i>	delayed IDLE mode entry interrupt flag, for Slave_TIMER
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the Master_TIMER and Slave_TIMER interrupt flag */
FlagStatus flag = RESET;

flag = shrtimer_timers_interrupt_flag_get(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_INT_FLAG_CMP0);
```

shrtimer_timers_interrupt_flag_clear

The description of shrtimer_timers_interrupt_flag_clear is shown as below:

Table 3-567. Function shrtimer_timers_interrupt_flag_clear

Function name	shrtimer_timers_interrupt_flag_clear
Function prototype	void shrtimer_timers_interrupt_flag_clear(uint32_t shrtimer_periph, uint32_t timer_id, uint32_t interrupt);
Function descriptions	clear the Master_TIMER and Slave_TIMER interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
timer_id	master timer and slave timer index
SHRTIMER_MASTER_	the master timer

TIMER	
SHRTIMER_SLAVE_TI MERx	the slave timer selection(x=0..4)
Input parameter{in}	
interrupt	the interrupt source
SHRTIMER_MT_ST_IN T _FLAG_CMPy(y=0..3)	compare y interrupt flag, for Master_TIMER and Slave_TIMER (y=0..3)
SHRTIMER_MT_ST_IN T_FLAG REP	repetition interrupt flag, for Master_TIMER and Slave_TIMER
SHRTIMER_MT_INT_F LAG_SYN	synchronization input interrupt flag, for Master_TIMER
SHRTIMER_MT_ST_IN T_FLAG_UPD	update interrupt flag, for Master_TIMER and Slave_TIMER
SHRTIMER_ST_INT_F LAG_CAPy(y=0,1)	capture y interrupt flag, for Slave_TIMER(y=0,1)
SHRTIMER_ST_INT_F LAG_CHyOA(y=0,1)	channel y output active interrupt flag, for Slave_TIMER(y=0,1)
SHRTIMER_ST_INT_F LAG_CHyONA(y=0,1)	channel y output inactive interrupt flag, for Slave_TIMER(y=0,1)
SHRTIMER_ST_INT_F LAG_CNRST	counter reset interrupt flag, for Slave_TIMER
SHRTIMER_ST_INT_F LAG_DLYIDLE	delayed IDLE mode entry interrupt flag, for Slave_TIMER
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the Master_TIMER and Slave_TIMER interrupt flag */

shrtimer_timers_interrupt_flag_clear(SHRTIMER0, SHRTIMER_MASTER_TIMER,
SHRTIMER_MT_ST_INT_FLAG_CMP0);
```

shrtimer_common_interrupt_enable

The description of shrtimer_common_interrupt_enable is shown as below:

Table 3-568. Function shrtimer_common_interrupt_enable

Function name	shrtimer_common_interrupt_enable
Function prototype	void shrtimer_common_interrupt_enable(uint32_t shrtimer_periph, uint32_t interrupt);
Function descriptions	enable SHRTIMER common interrupt

Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
interrupt	the interrupt source
SHRTIMER_INT_FLTy(<i>y=0..3)</i>	fault y interrupt (<i>y</i> =0..3)
SHRTIMER_INT_SYSF <i>LT</i>	system fault interrupt
SHRTIMER_INT_DLLC <i>AL</i>	DLL calibration completed interrupt
SHRTIMER_INT_BMP <i>ER</i>	bunch mode period interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SHRTIMER common interrupt */
shrtimer_common_interrupt_enable(SHRTIMER0, SHRTIMER_INT_FLT0);
```

shrtimer_common_interrupt_disable

The description of shrtimer_common_interrupt_disable is shown as below:

Table 3-569. Function shrtimer_common_interrupt_disable

Function name	shrtimer_common_interrupt_disable
Function prototype	void shrtimer_common_interrupt_disable(uint32_t shrtimer_periph, uint32_t interrupt);
Function descriptions	disable SHRTIMER common interrupt
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
interrupt	the interrupt source
SHRTIMER_INT_FLTy(<i>y=0..3)</i>	fault y interrupt (<i>y</i> =0..3)
SHRTIMER_INT_SYSF	system fault interrupt

<i>LT</i>	
<i>SHRTIMER_INT_DLLC</i> <i>AL</i>	DLL calibration completed interrupt
<i>SHRTIMER_INT_BMP</i> <i>ER</i>	bunch mode period interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SHRTIMER common interrupt */
shrtimer_common_interrupt_disable(SHRTIMER0, SHRTIMER_INT_FLT0);
```

shrtimer_common_interrupt_flag_get

The description of shrtimer_common_interrupt_flag_get is shown as below:

Table 3-570. Function shrtimer_common_interrupt_flag_get

Function name	shrtimer_common_interrupt_flag_get
Function prototype	FlagStatus shrtimer_common_interrupt_flag_get(uint32_t shrtimer_periph, uint32_t interrupt);
Function descriptions	get the common interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
<i>shrtimer_periph</i>	SHRTIMER peripheral
<i>SHRTIMERx(x=0)</i>	SHRTIMER selection
Input parameter{in}	
<i>interrupt</i>	the interrupt source
<i>SHRTIMER_INT</i> <i>_FLAG_FLTy(y=0..3)</i>	fault y interrupt flag (y=0..3)
<i>SHRTIMER_INT</i> <i>_FLAG_SYSFLT</i>	system fault interrupt flag
<i>SHRTIMER_INT</i> <i>_FLAG_DLLCAL</i>	DLL calibration completed interrupt flag
<i>SHRTIMER_INT</i> <i>_FLAG_BMPER</i>	bunch mode period interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```

/* get the common interrupt flag */

FlagStatus flag = RESET;

flag = shrtimer_common_interrupt_flag_get(SHRTIMER0, SHRTIMER_INT_FLAG_FLT0);
  
```

shrtimer_common_interrupt_flag_clear

The description of shrtimer_common_interrupt_flag_clear is shown as below:

Table 3-571. Function shrtimer_common_interrupt_flag_clear

Function name	shrtimer_common_interrupt_flag_clear
Function prototype	void shrtimer_common_interrupt_flag_clear(uint32_t shrtimer_periph, uint32_t interrupt);
Function descriptions	clear the common interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_periph	SHRTIMER peripheral
SHRTIMERx(x=0)	SHRTIMER selection
Input parameter{in}	
interrupt	the interrupt source
SHRTIMER_INT_FLAG_FLT_y(y=0..3)	fault y interrupt flag (y=0..3)
SHRTIMER_INT_FLAG_SYSFLT	system fault interrupt flag
SHRTIMER_INT_FLAG_DLLCAL	DLL calibration completed interrupt flag
SHRTIMER_INT_FLAG_BMPER	bunch mode period interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* clear the common interrupt flag */

shrtimer_common_interrupt_flag_clear(SHRTIMER0, SHRTIMER_INT_FLAG_FLT0);
  
```

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 3-572. I2C Registers

Registers	Descriptions
I2C_CTL0	Control register 0
I2C_CTL1	Control register 1
I2C_SADDR0	Slave address register 0
I2C_SADDR1	Slave address register 1
I2C_DATA	Transfer buffer register
I2C_STAT0	Transfer status register 0
I2C_STAT1	Transfer status register 1
I2C_CKCFG	Clock configure register
I2C_RT	Rise time register
I2C_SAMCS	SAM control and status register
I2C_CTL2	Control register 2
I2C_CS	Control and status register
I2C_STATC	Status clear register
I2C2_CTL0	Control register 0
I2C2_CTL1	Control register 1
I2C2_SADDR0	Slave address register 0
I2C2_SADDR1	Slave address register 1
I2C2_TIMING	Timing register
I2C2_TIMEOUT	Timeout register
I2C2_STAT	Status register
I2C2_STATC	Status clear register
I2C2_PEC	PEC register
I2C2_RDATA	Receive data register
I2C2_TDATA	Transmit data register

3.18.2. Descriptions of Peripheral functions

I2C firmware functions are listed in the table shown as below:

Table 3-573. I2C firmware function

Function name	Function description
i2c_deinit	reset I2C
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

Function name	Function description
i2c_slave_response_to_gcall_enable	enable the response to a general call
i2c_slave_response_to_gcall_disable	disable the response to a general call
i2c_stretch_scl_low_enable	enable to stretch SCL low when data is not ready in slave mode
i2c_stretch_scl_low_disable	disable to stretch SCL low when data is not ready in slave mode
i2c_data_transmit	I2C transmit data function
i2c_data_receive	I2C receive data function
i2c_pec_transfer	I2C transfers PEC value
i2c_pec_enable	enable I2C PEC calculation
i2c_pec_disable	disable I2C PEC calculation
i2c_pec_value_get	get packet error checking value
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	dual-address mode switch
i2c_dualaddr_disable	disable dual-address mode
i2c_dma_config	configure I2C DMA mode
i2c_dma_last_transfer_config	configure whether next DMA EOT is DMA last transfer or not
i2c_software_reset_config	configure software reset I2C
i2c_smbus_alert_config	configure I2C alert through SMBA pin
i2c_smbus_arp_config	configure I2C ARP protocol in SMBus
i2c_sam_enable	enable SAM_V interface
i2c_sam_disable	disable SAM_V interface
i2c_sam_timeout_enable	enable SAM_V interface timeout detect
i2c_sam_timeout_disable	disable SAM_V interface timeout detect
i2c_start_early_termination_mode_config	configure I2C start early termination mode
i2c_timeout_calculation_enable	enable i2c timeout calculation
i2c_timeout_calculation_disable	disable i2c timeout calculation
i2c_record_received_slave_address_enable	enable i2c record the received slave address to the transfer buffer register
i2c_record_received_slave_address_disable	disable i2c record the received slave address to the transfer buffer register
i2c_address_bit_compare_config	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
i2c_status_clear_enable	enable i2c status register clear
i2c_status_clear_disable	disable i2c status register clear

Function name	Function description
i2c_status_bit_clear	clear i2c status register bit
i2c_flag_get	get I2C flag status
i2c_flag_clear	clear I2C flag status
i2c_interrupt_enable	enable I2C interrupt
i2c_interrupt_disable	disable I2C interrupt
i2c_interrupt_flag_get	get I2C interrupt flag
i2c_interrupt_flag_clear	clear I2C interrupt flag
i2c_timing_config	configure the timing parameters
i2c_digital_noise_filter_config	configure digital noise filter
i2c_analog_noise_filter_enable	enable analog noise filter
i2c_analog_noise_filter_disable	disable analog noise filter
i2c_wakeup_from_deepsleep_enable	enable wakeup from Deep-sleep mode
i2c_wakeup_from_deepsleep_disable	disable wakeup from Deep-sleep mode
i2c_master_clock_config	configure the SCL high and low period of clock in master mode
i2c2_master_addressing	configure i2c slave addresss and transfer direction in master mode
i2c_address10_header_enable	10-bit address header executes read direction only in master receive mode
i2c_address10_header_disable	10-bit address header executes complete sequence in master receive mode
i2c_address10_enable	enable 10-bit addressing mode in master mode
i2c_address10_disable	disable 10-bit addressing mode in master mode
i2c_automatic_end_enable	enable I2C automatic end mode in master mode
i2c_automatic_end_disable	disable I2C automatic end mode in master mode
i2c_address_config	configure i2c slave address
i2c_address_disable	disable i2c address in slave mode
i2c_second_address_config	configure i2c second slave address
i2c_second_address_disable	disable i2c second address in slave mode
i2c_recevied_address_get	get received match address in slave mode
i2c_slave_byte_control_enable	enable slave byte control
i2c_slave_byte_control_disable	disable slave byte control
i2c_nack_enable	generate a NACK in slave mode
i2c_nack_disable	generate an ACK in slave mode
i2c_reload_enable	enable I2C reload mode
i2c_reload_disable	disable I2C reload mode
i2c_transfer_byte_number_config	configure number of bytes to be transferred
i2c2_dma_enable	enable I2C DMA for transmission or reception
i2c2_dma_disable	disable I2C DMA for transmission or reception
i2c_smbus_alert_enable	enable SMBus Alert
i2c_smbus_alert_disable	disable SMBus Alert

Function name	Function description
i2c_smbus_default_addr_enable	enable SMBus device default address
i2c_smbus_default_addr_disable	disable SMBus device default address
i2c_smbus_host_addr_enable	enable SMBus Host address
i2c_smbus_host_addr_disable	disable SMBus Host address
i2c_extented_clock_timeout_enable	enable extended clock timeout detection
i2c_extented_clock_timeout_disable	disable extended clock timeout detection
i2c_clock_timeout_enable	enable clock timeout detection
i2c_clock_timeout_disable	disable clock timeout detection
i2c_bus_timeout_b_config	configure bus timeout B
i2c_bus_timeout_a_config	configure bus timeout A
i2c_idle_clock_timeout_config	configure idle clock timeout detection
i2c2_flag_get	get I2C flag status
i2c2_flag_clear	clear I2C flag status
i2c2_interrupt_enable	enable I2C interrupt
i2c2_interrupt_disable	disable I2C interrupt
i2c2_interrupt_flag_get	get I2C interrupt flag
i2c2_interrupt_flag_clear	clear I2C interrupt flag

Enum i2c_flag_enum

Table 3-574. i2c_flag_enum

Member name	Function description
I2C_FLAG_SBSEND	start condition send out
I2C_FLAG_ADDSEND	address is sent in master mode or received and matches in slave mode
I2C_FLAG_BTC	byte transmission finishes
I2C_FLAG_ADD10SEND	header of 10-bit address is sent in master mode
I2C_FLAG_STPDET	stop condition detected in slave mode
I2C_FLAG_RBNE	I2C_DATA is not Empty during receiving
I2C_FLAG_TBE	I2C_DATA is empty during transmitting
I2C_FLAG_BERR	a bus error occurs indication a unexpected start or stop condition on I2C bus
I2C_FLAG_LOSTARB	arbitration lost in master mode
I2C_FLAG_AERR	acknowledge error
I2C_FLAG_OUERR	overrun or underrun situation occurs in slave mode
I2C_FLAG_PECERR	PEC error when receiving data
I2C_FLAG_SMBTO	timeout signal in SMBus mode
I2C_FLAG_SMBALT	SMBus alert status
I2C_FLAG_MASTER	a flag indicating whether I2C block is in master or slave mode
I2C_FLAG_I2CBSY	busy flag
I2C_FLAG_TR	whether the I2C is a transmitter or a receiver
I2C_FLAG_RXGC	general call address (00h) received

Member name	Function description
I2C_FLAG_DEFSMB	default address of SMBus device
I2C_FLAG_HSTSMB	SMBus host header detected in slave mode
I2C_FLAG_DUMOD	dual flag in slave mode indicating which address is matched in dual-address mode
I2C_FLAG_TFF	txframe fall flag
I2C_FLAG_TFR	txframe rise flag
I2C_FLAG_RFF	rxframe fall flag
I2C_FLAG_RFR	rxframe rise flag
I2C_FLAG_STLO	start lost flag
I2C_FLAG_STPSEND	stop condition sent flag

Enum i2c_interrupt_enum

Table 3-575. i2c_interrupt_enum

Member name	Function description
I2C_INT_ERR	error interrupt disable
I2C_INT_EV	event interrupt disable
I2C_INT_BUF	buffer interrupt disable
I2C_INT_TFF	txframe fall interrupt enable
I2C_INT_TFR	txframe rise interrupt enable
I2C_INT_RFF	rxframe fall interrupt enable
I2C_INT_RFR	rxframe rise interrupt enable
I2C_INT_STLO	start lost interrupt enable
I2C_INT_STPSEND	stop condition sent interrupt enable

Enum i2c_interrupt_flag_enum

Table 3-576. i2c_interrupt_flag_enum

Member name	Function description
I2C_INT_FLAG_SBSEND	start condition sent out in master mode interrupt flag
I2C_INT_FLAG_ADDSEND	address is sent in master mode or received and matches in slave mode interrupt flag
I2C_INT_FLAG_BTC	byte transmission finishes
I2C_INT_FLAG_ADD10SEND	header of 10-bit address is sent in master mode interrupt flag
I2C_INT_FLAG_STPDET	stop condition detected in slave mode interrupt flag
I2C_INT_FLAG_RBNE	I2C_DATA is not Empty during receiving interrupt flag
I2C_INT_FLAG_TBE	I2C_DATA is empty during transmitting interrupt flag
I2C_INT_FLAG_BERR	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
I2C_INT_FLAG_LOSTARB	arbitration lost in master mode interrupt flag
I2C_INT_FLAG_AERR	acknowledge error interrupt flag
I2C_INT_FLAG_OUERR	over-run or under-run situation occurs in slave mode interrupt flag

Member name	Function description
I2C_INT_FLAG_PECERR	PEC error when receiving data interrupt flag
I2C_INT_FLAG_SMBTO	timeout signal in SMBus mode interrupt flag
I2C_INT_FLAG_SMBALT	SMBus Alert status interrupt flag
I2C_INT_FLAG_TFF	txframe fall interrupt flag
I2C_INT_FLAG_TFR	txframe rise interrupt flag
I2C_INT_FLAG_RFF	rxframe fall interrupt flag
I2C_INT_FLAG_RFR	rxframe rise interrupt flag
I2C_INT_FLAG_STLO	start lost interrupt flag
I2C_INT_FLAG_STPSEND	stop condition sent interrupt flag

Enum i2c2_interrupt_flag_enum

Table 3-577. i2c2_interrupt_flag_enum

Member name	Function description
I2C2_INT_FLAG_TI	transmit interrupt flag
I2C2_INT_FLAG_RBNE	I2C_RDATA is not empty during receiving interrupt flag
I2C2_INT_FLAG_ADDSEND	address received matches in slave mode interrupt flag
I2C2_INT_FLAG_NACK	not acknowledge interrupt flag
I2C2_INT_FLAG_STPDET	stop condition detected in slave mode interrupt flag
I2C2_INT_FLAG_TC	transfer complete in master mode interrupt flag
I2C2_INT_FLAG_TCR	transfer complete reload interrupt flag
I2C2_INT_FLAG_BERR	bus error interrupt flag
I2C2_INT_FLAG_LOSTARB	arbitration lost interrupt flag
I2C2_INT_FLAG_OUERR	overrun/underrun error in slave mode interrupt flag
I2C2_INT_FLAG_PECERR	PEC error interrupt flag
I2C2_INT_FLAG_TIMEOUT	timeout interrupt flag
I2C2_INT_FLAG_SMBALT	SMBus Alert interrupt flag

i2c_deinit

The description of i2c_deinit is shown as below:

Table 3-578. Function i2c_deinit

Function name	i2c_deinit
Function prototype	void i2c_deinit(uint32_t i2c_periph);
Function descriptions	reset I2C
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* reset I2C0 */

i2c_deinit (I2C0);
```

i2c_enable

The description of i2c_enable is shown as below:

Table 3-579. Function i2c_enable

Function name	i2c_enable
Function prototype	void i2c_enable(uint32_t i2c_periph);
Function descriptions	enable I2C
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 */

i2c_enable (I2C0);
```

i2c_disable

The description of i2c_disable is shown as below:

Table 3-580. Function i2c_disable

Function name	i2c_disable
Function prototype	void i2c_disable(uint32_t i2c_periph);
Function descriptions	disable I2C
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable I2C0 */
i2c_disable (I2C0);
```

i2c_start_on_bus

The description of i2c_start_on_bus is shown as below:

Table 3-581. Function i2c_start_on_bus

Function name	i2c_start_on_bus
Function prototype	void i2c_start_on_bus(uint32_t i2c_periph);
Function descriptions	generate a START condition on I2C bus
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 send a start condition to I2C bus */
i2c_start_on_bus (I2C0);
```

i2c_stop_on_bus

The description of i2c_stop_on_bus is shown as below:

Table 3-582. Function i2c_stop_on_bus

Function name	i2c_stop_on_bus
Function prototype	void i2c_stop_on_bus(uint32_t i2c_periph);
Function descriptions	generate a STOP condition on I2C bus
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* I2C0 generate a STOP condition to I2C bus */
```

```
i2c_stop_on_bus (I2C0);
```

i2c_slave_response_to_gcall_enable

The description of i2c_slave_response_to_gcall_enable is shown as below:

Table 3-583. Function i2c_slave_response_to_gcall_enable

Function name	i2c_slave_response_to_gcall_enable
Function prototype	void i2c_slave_response_to_gcall_enable(uint32_t i2c_periph);
Function descriptions	enable the response to a general call
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the response to a general call */
```

```
i2c_slave_response_to_gcall_enable (I2C0);
```

i2c_slave_response_to_gcall_disable

The description of i2c_slave_response_to_gcall_disable is shown as below:

Table 3-584. Function i2c_slave_response_to_gcall_disable

Function name	i2c_slave_response_to_gcall_disable
Function prototype	void i2c_slave_response_to_gcall_disable(uint32_t i2c_periph);
Function descriptions	disable the response to a general call
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable the response to a general call */

i2c_slave_response_to_gcall_disable (I2C0);
```

i2c_stretch_scl_low_enable

The description of i2c_stretch_scl_low_enable is shown as below:

Table 3-585. Function i2c_stretch_scl_low_enable

Function name	i2c_stretch_scl_low_enable
Function prototype	void i2c_stretch_scl_low_enable(uint32_t i2c_periph);
Function descriptions	enable to stretch SCL low when data is not ready in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable to stretch SCL low when data is not ready in slave mode */

i2c_stretch_scl_low_enable (I2C0);
```

i2c_stretch_scl_low_disable

The description of i2c_stretch_scl_low_disable is shown as below:

Table 3-586. Function i2c_stretch_scl_low_disable

Function name	i2c_stretch_scl_low_disable
Function prototype	void i2c_stretch_scl_low_disable(uint32_t i2c_periph);
Function descriptions	disable to stretch SCL low when data is not ready in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable to stretch SCL low when data is not ready in slave mode */
i2c_stretch_scl_low_disable (I2C0);
```

i2c_data_transmit

The description of i2c_data_transmit is shown as below:

Table 3-587. Function i2c_data_transmit

Function name	i2c_data_transmit
Function prototype	void i2c_data_transmit(uint32_t i2c_periph, uint32_t data);
Function descriptions	I2C transmit data function
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
data	transmit data
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 transmit data */
i2c_data_transmit (I2C0, 0x55);
```

i2c_data_receive

The description of i2c_data_receive is shown as below:

Table 3-588. 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	
uint32_t	0x00..0xFF

Example:

```
/* I2C0 receive data */

uint32_t i2c_receiver;

i2c_receiver = i2c_data_receive(I2C0);
```

i2c_pec_transfer

The description of i2c_pec_transfer is shown as below:

Table 3-589. Function i2c_pec_transfer

Function name	i2c_pec_transfer	
Function prototype	void i2c_pec_transfer(uint32_t i2c_periph);	
Function descriptions	I2C transfers PEC value	
Precondition	-	
The called functions	-	
Input parameter{in}		
i2c_periph	I2C peripheral	
I2Cx	(x=0,1,2)	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

```
/* I2C transfers PEC value */

i2c_pec_transfer (I2C0);
```

i2c_pec_enable

The description of i2c_pec_enable is shown as below:

Table 3-590. Function i2c_pec_enable

Function name	i2c_pec_enable	
Function prototype	void i2c_pec_enable(uint32_t i2c_periph);	
Function descriptions	enable I2C PEC calculation	
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 I2C PEC calculation */

i2c_pec_enable (I2C0);
```

i2c_pec_disable

The description of **i2c_pec_disable** is shown as below:

Table 3-591. Function i2c_pec_disable

Function name	i2c_pec_disable
Function prototype	void i2c_pec_disable(uint32_t i2c_periph);
Function descriptions	disable I2C PEC calculation
Precondition	-
The called functions	-
Input parameter{in}	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C PEC calculation */

i2c_pec_disable (I2C0);
```

i2c_pec_value_get

The description of **i2c_pec_value_get** is shown as below:

Table 3-592. Function i2c_pec_value_get

Function name	i2c_pec_value_get
Function prototype	uint32_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	
uint32_t	PEC value

Example:

```
/* I2C0 get packet error checking value */

uint32_t pec_value;

pec_value = i2c_pec_value_get (I2C0);
```

i2c_clock_config

The description of **i2c_clock_config** is shown as below:

Table 3-593. Function i2c_clock_config

Function name	i2c_clock_config
Function prototype	void i2c_clock_config(uint32_t i2c_periph, uint32_t clkspeed, uint32_t dutycyc);
Function descriptions	I2C clock configure
Precondition	-
The called functions	rcu_clock_freq_get
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
clkspeed	i2c clock speed
Input parameter{in}	
dutycyc	duty cycle in fast mode
<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 3-594. 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}	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
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 3-595. Function i2c_smbus_type_config

Function name	i2c_smbus_type_config
Function prototype	void i2c_smbus_type_config(uint32_t i2c_periph, uint32_t type);
Function descriptions	SMBus type selection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
type	Device or host
I2C_SMBUS_DEVICE	device
I2C_SMBUS_HOST	host
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config I2C0 as SMBUS host type*/
i2c_smbus_type_config (I2C0, I2C_SMBUS_HOST);
```

i2c_ack_config

The description of i2c_ack_config is shown as below:

Table 3-596. 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
I2Cx	(x=0,1)
Input parameter{in}	
ack	I2C peripheral
I2C_ACK_ENABLE	ACK will be sent
I2C_ACK_DISABLE	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 3-597. Function i2c_ackpos_config

Function name	i2c_ackpos_config
Function prototype	void i2c_ackpos_config(uint32_t i2c_periph, uint32_t pos);
Function descriptions	I2C POAP position configure
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
pos	ACK position
I2C_ACKPOS_CURRENT	whether to send ACK or not for the current
I2C_ACKPOS_NEXT	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 3-598. 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)
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 3-599. Function i2c_dualaddr_enable

Function name	i2c_dualaddr_enable
Function prototype	void i2c_dualaddr_enable(uint32_t i2c_periph, uint32_t dualaddr);
Function descriptions	enable dual-address mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
addr	the second address in dual-address mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 dual-address */
i2c_dualaddr_enable (I2C0, 0x82);
```

i2c_dualaddr_disable

The description of i2c_dualaddr_disable is shown as below:

Table 3-600. 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
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable dual-address mode */

i2c_dualaddr_disable (I2C0);
```

i2c_dma_config

The description of i2c_dma_config is shown as below:

Table 3-601. Function i2c_dma_config

Function name	i2c_dma_config
Function prototype	void i2c_dma_config (uint32_t i2c_periph, uint32_t dmastate);
Function descriptions	configure I2C DMA mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
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_config (I2C0, I2C_DMA_ON);
```

i2c_dma_last_transfer_config

The description of i2c_dma_last_transfer_config is shown as below:

Table 3-602. 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	flag indicating DMA last transfer
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
dmalast	next DMA EOT is the last transfer or not
I2C_DMALST_ON	next DMA EOT is the last transfer
I2C_DMALST_OFF	next DMA EOT is not the last transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* next DMA EOT is the last transfer */

i2c_dma_last_transfer_enable (I2C0, I2C_DMALST_ON);
```

i2c_software_reset_config

The description of i2c_software_reset_config is shown as below:

Table 3-603. Function i2c_software_reset_config

Function name	i2c_software_reset_config
Function prototype	void i2c_software_reset_config(uint32_t i2c_periph, uint32_t sreset);
Function descriptions	configure software reset I2C
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)

Input parameter{in}	
sreset	under reset or not
<i>I2C_SRESET_SET</i>	I2C is under reset
<i>I2C_SRESET_RESET</i>	I2C is not under reset
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* software reset I2C0 */
i2c_software_reset_config (I2C0, I2C_SRESET_SET);
```

i2c_smbus_alert_config

The description of i2c_smbus_alert_config is shown as below:

Table 3-604. Function i2c_smbus_alert_config

Function name	i2c_smbus_alert_config	
Function prototype	void i2c_smbus_alert_config(uint32_t i2c_periph, uint32_t smbuspara);	
Function descriptions	configure I2C alert through SMBA pin	
Precondition	-	
The called functions	-	
Input parameter{in}		
i2c_periph	I2C peripheral	
<i>I2Cx</i>	(x=0,1)	
Input parameter{in}		
smbuspara	issue alert through SMBA pin or not	
<i>I2C_SALTSEND_ENAB</i> <i>LE</i>	issue alert through SMBA pin	
<i>I2C_SALTSEND_DISA</i> <i>BLE</i>	not issue alert through SMBA pin	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

```
/* I2C0 issue alert through SMBA pin enable*/
i2c_smbus_alert_config (I2C0, I2C_SALTSEND_ENABLE);
```

i2c_smbus_arp_config

The description of i2c_smbus_arp_config is shown as below:

Table 3-605. Function i2c_smbus_arp_config

Function name	i2c_smbus_arp_config
Function prototype	void i2c_smbus_arp_config(uint32_t i2c_periph, uint32_t arpstate);
Function descriptions	configure I2C ARP protocol in SMBus
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
arpstate	ARP protocol in SMBus switch
I2C_ARP_ENABLE	enable ARP
I2C_ARP_DISABLE	disable ARP
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 ARP protocol in SMBus switch */

i2c_smbus_arp_config (I2C0, I2C_ARP_ENABLE);
```

i2c_sam_enable

The description of i2c_sam_enable is shown as below:

Table 3-606. Function i2c_sam_enable

Function name	i2c_sam_enable
Function prototype	void i2c_sam_enable(uint32_t i2c_periph);
Function descriptions	enable SAM_V interface
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SAM_V interface */

i2c_sam_enable (I2C0);
```

i2c_sam_disable

The description of i2c_sam_disable is shown as below:

Table 3-607. Function i2c_sam_disable

Function name	i2c_sam_disable
Function prototype	void i2c_sam_disable(uint32_t i2c_periph);
Function descriptions	disable SAM_V interface
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SAM_V interface */

i2c_sam_disable (I2C0);
```

i2c_sam_timeout_enable

The description of i2c_sam_timeout_enable is shown as below:

Table 3-608. Function i2c_sam_timeout_enable

Function name	i2c_sam_timeout_enable
Function prototype	void i2c_sam_timeout_enable(uint32_t i2c_periph);
Function descriptions	enable SAM_V interface timeout detect
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SAM_V interface timeout detect */

i2c_sam_timeout_enable (I2C0);
```

i2c_sam_timeout_disable

The description of i2c_sam_timeout_disable is shown as below:

Table 3-609. Function i2c_sam_timeout_disable

Function name	i2c_sam_timeout_disable
Function prototype	void i2c_sam_timeout_disable(uint32_t i2c_periph);
Function descriptions	disable SAM_V interface timeout detect
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SAM_V interface timeout detect */

i2c_sam_timeout_disable (I2C0);
```

i2c_start_early_termination_mode_config

The description of i2c_start_early_termination_mode_config is shown as below:

Table 3-610. Function i2c_start_early_termination_mode_config

Function name	i2c_start_early_termination_mode_config
Function prototype	void i2c_start_early_termination_mode_config(uint32_t i2c_periph, uint32_t mode);
Function descriptions	configure I2C start early termination mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
mode	I2C start early termination mode
STANDARD_I2C_PRO	do as the standard i2c protocol

<i>TOCOL_MODE</i>	
<i>ARBITRATION_LOST_MODE</i>	do the same thing as arbitration lost
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure I2C start early termination mode */

i2c_start_early_termination_mode_config (I2C0, ARBITRATION_LOST_MODE);
```

i2c_timeout_calculation_enable

The description of *i2c_timeout_calculation_enable* is shown as below:

Table 3-611. Function i2c_timeout_calculation_enable

Function name	i2c_timeout_calculation_enable
Function prototype	void i2c_timeout_calculation_enable(uint32_t i2c_periph);
Function descriptions	enable i2c timeout calculation
Precondition	-
The called functions	-
Input parameter{in}	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable i2c timeout calculation */

i2c_timeout_calculation_enable (I2C0);
```

i2c_timeout_calculation_disable

The description of *i2c_timeout_calculation_disable* is shown as below:

Table 3-612. Function i2c_timeout_calculation_disable

Function name	i2c_timeout_calculation_disable
Function prototype	void i2c_timeout_calculation_disable(uint32_t i2c_periph);
Function descriptions	disable i2c timeout calculation
Precondition	-
The called functions	-

Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable i2c timeout calculation */

i2c_timeout_calculation_disable (I2C0);
```

i2c_record_received_slave_address_enable

The description of i2c_record_received_slave_address_enable is shown as below:

Table 3-613. Function i2c_record_received_slave_address_enable

Function name	i2c_record_received_slave_address_enable
Function prototype	void i2c_record_received_slave_address_enable(uint32_t i2c_periph);
Function descriptions	enable i2c record the received slave address to the transfer buffer register
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable i2c record the received slave address to the transfer buffer register */

i2c_record_received_slave_address_enable (I2C0);
```

i2c_record_received_slave_address_disable

The description of i2c_record_received_slave_address_disable is shown as below:

Table 3-614. Function i2c_record_received_slave_address_disable

Function name	i2c_record_received_slave_address_disable
Function prototype	void i2c_record_received_slave_address_disable(uint32_t i2c_periph);
Function descriptions	disable i2c record the received slave address to the transfer buffer register
Precondition	-
The called functions	-

Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable i2c record the received slave address to the transfer buffer register */

i2c_record_received_slave_address_disable (I2C0);
```

i2c_address_bit_compare_config

The description of i2c_address_bit_compare_config is shown as below:

Table 3-615. Function i2c_address_bit_compare_config

Function name	i2c_address_bit_compare_config
Function prototype	void i2c_address_bit_compare_config(uint32_t i2c_periph, uint16_t compare_bits);
Function descriptions	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
mode	the bits need to compare
ADDRESS_BIT1_COM_PARE	address bit1 needs compare
ADDRESS_BIT2_COM_PARE	address bit2 needs compare
ADDRESS_BIT3_COM_PARE	address bit3 needs compare
ADDRESS_BIT4_COM_PARE	address bit4 needs compare
ADDRESS_BIT5_COM_PARE	address bit5 needs compare
ADDRESS_BIT6_COM_PARE	address bit6 needs compare
ADDRESS_BIT7_COM_PARE	address bit7 needs compare

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* define which bits of ADDRESS[7:1] need to compare with the incoming address byte */

i2c_address_bit_compare_config (I2C0, ADDRESS_BIT7_COMPARE);
```

i2c_status_clear_enable

The description of i2c_status_clear_enable is shown as below:

Table 3-616. Function i2c_status_clear_enable

Function name	i2c_status_clear_enable
Function prototype	void i2c_status_clear_enable(uint32_t i2c_periph);
Function descriptions	enable i2c status register clear
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable i2c status register clear */

i2c_status_clear_enable (I2C0);
```

i2c_status_clear_disable

The description of i2c_status_clear_disable is shown as below:

Table 3-617. Function i2c_status_clear_disable

Function name	i2c_status_clear_disable
Function prototype	void i2c_status_clear_disable(uint32_t i2c_periph);
Function descriptions	disable i2c status register clear
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable i2c status register clear */

i2c_status_clear_disable (I2C0);
```

i2c_status_bit_clear

The description of i2c_status_bit_clear is shown as below:

Table 3-618. Function i2c_start_early_termination_mode_config

Function name	i2c_status_bit_clear
Function prototype	void i2c_status_bit_clear(uint32_t i2c_periph, uint32_t clear_bit);
Function descriptions	clear i2c status register bit
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
clear_bit	which bit needs to clear
CLEAR_STPDET	clear STPDET bit in I2C_STAT0
CLEAR_ADD10SEND	clear ADD10SEND bit in I2C_STAT0
CLEAR_BTC	clear BTC bit in I2C_STAT0
CLEAR_ADDSEND	clear ADDSEND bit in I2C_STAT0
CLEAR_SBSEND	clear SBSEND bit in I2C_STAT0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear i2c status register bit */

i2c_status_bit_clear (I2C0, CLEAR_ADDSEND);
```

i2c_flag_get

The description of i2c_flag_get is shown as below:

Table 3-619. Function i2c_flag_get

Function name	i2c_flag_get
----------------------	--------------

Function prototype	FlagStatus i2c_flag_get(uint32_t i2c_periph, i2c_flag_enum flag);
Function descriptions	get I2C flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
flag	specify get which flag, refer to Table 3-574. i2c_flag_enum .
I2C_FLAG_SBSEND	start condition send out
I2C_FLAG_ADDSEND	address is sent in master mode or received and matches in slave mode
I2C_FLAG_BTC	byte transmission finishes
I2C_FLAG_ADD10SEN_D	header of 10-bit address is sent in master mode
I2C_FLAG_STPDET	stop condition detected in slave mode
I2C_FLAG_RBNE	I2C_DATA is not Empty during receiving
I2C_FLAG_TBE	I2C_DATA is empty during transmitting
I2C_FLAG_BERR	a bus error occurs indication a unexpected start or stop condition on I2C bus
I2C_FLAG_LOSTARB	arbitration lost in master mode
I2C_FLAG_AERR	acknowledge error
I2C_FLAG_OVERRR	overrun or underrun situation occurs in slave mode
I2C_FLAG_PECERR	PEC error when receiving data
I2C_FLAG_SMBTO	timeout signal in SMBus mode
I2C_FLAG_SMBALT	SMBus alert status
I2C_FLAG_MASTER	a flag indicating whether I2C block is in master or slave mode
I2C_FLAG_I2CBSY	busy flag
I2C_FLAG_TR	whether the I2C is a transmitter or a receiver
I2C_FLAG_RXGC	general call address (00h) received
I2C_FLAG_DEFSMB	default address of SMBus device
I2C_FLAG_HSTSMB	SMBus host header detected in slave mode
I2C_FLAG_DUMOD	dual flag in slave mode indicating which address is matched in dual-address mode
I2C_FLAG_TFF	txframe fall flag
I2C_FLAG_TFR	txframe rise flag
I2C_FLAG_RFF	rxframe fall flag
I2C_FLAG_RFR	rxframe rise flag
I2C_FLAG_STLO	start lost flag
I2C_FLAG_STPSEND	stop condition sent flag
Output parameter{out}	
-	-
Return value	

FlagStatus	SET / RESET
------------	-------------

Example:

```
/* check whether start condition send out */

FlagStatus flag_state = RESET;

flag_state = i2c_flag_get (I2C0, I2C_FLAG_SBSEND);
```

i2c_flag_clear

The description of i2c_flag_clear is shown as below:

Table 3-620. Function i2c_flag_clear

Function name	i2c_flag_clear
Function prototype	void i2c_flag_clear(uint32_t i2c_periph, i2c_flag_enum flag);
Function descriptions	clear I2C flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
flag	flag type, refer to Table 3-574. i2c_flag_enum .
I2C_FLAG_SMBALT	SMBus Alert status
I2C_FLAG_SMBTO	timeout signal in SMBus mode
I2C_FLAG_PECERR	PEC error when receiving data
I2C_FLAG_OUERR	over-run or under-run situation occurs in slave mode
I2C_FLAG_AERR	acknowledge error
I2C_FLAG_LOSTARB	arbitration lost in master mode
I2C_FLAG_BERR	a bus error
I2C_FLAG_ADDSEND	cleared by reading I2C_STAT0 and reading I2C_STAT1
I2C_FLAG_TFF	txframe fall flag
I2C_FLAG_TFR	txframe rise flag
I2C_FLAG_RFF	rxframe fall flag
I2C_FLAG_RFR	rxframe rise flag
I2C_FLAG_STLO	start lost flag
I2C_FLAG_STPSEND	stop condition sent flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear a bus error flag*/
```

```
i2c_flag_clear (I2C0, I2C_FLAG_BERR);
```

i2c_interrupt_enable

The description of i2c_interrupt_enable is shown as below:

Table 3-621. Function i2c_interrupt_enable

Function name	i2c_interrupt_enable
Function prototype	void i2c_interrupt_enable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
Function descriptions	enable I2C interrupt
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
interrupt	interrupt type, refer to Table 3-575. i2c_interrupt_enum .
I2C_INT_ERR	error interrupt enable
I2C_INT_EV	event interrupt enable
I2C_INT_BUF	buffer interrupt enable
I2C_INT_TFF	txframe fall interrupt enable
I2C_INT_TFR	txframe rise interrupt enable
I2C_INT_RFF	rxframe fall interrupt enable
I2C_INT_RFR	rxframe rise interrupt enable
I2C_INT_STLO	start lost interrupt enable
I2C_INT_STPSEND	stop condition sent 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 3-622. 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
I2Cx	(x=0,1)
Input parameter{in}	
interrupt	interrupt type, refer to Table 3-575. i2c_interrupt_enum .
I2C_INT_ERR	error interrupt disable
I2C_INT_EV	event interrupt disable
I2C_INT_BUF	buffer interrupt disable
I2C_INT_TFF	txframe fall interrupt enable
I2C_INT_TFR	txframe rise interrupt enable
I2C_INT_RFF	rxframe fall interrupt enable
I2C_INT_RFR	rxframe rise interrupt enable
I2C_INT_STLO	start lost interrupt enable
I2C_INT_STPSEND	stop condition sent interrupt enable
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 3-623. 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	get I2C interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
int_flag	interrupt flag, refer to Table 3-576. i2c_interrupt_flag_enum .
I2C_INT_FLAG_SBSE ND	start condition sent out in master mode interrupt flag

<i>I2C_INT_FLAG_ADDS END</i>	address is sent in master mode or received and matches in slave mode interrupt flag
<i>I2C_INT_FLAG_BTC</i>	byte transmission finishes
<i>I2C_INT_FLAG_ADD10 SEND</i>	header of 10-bit address is sent in master mode interrupt flag
<i>I2C_INT_FLAG_STPD ET</i>	stop condition detected in slave mode interrupt flag
<i>I2C_INT_FLAG_RBNE</i>	I2C_DATA is not Empty during receiving interrupt flag
<i>I2C_INT_FLAG_TBE</i>	I2C_DATA is empty during transmitting interrupt flag
<i>I2C_INT_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
<i>I2C_INT_FLAG_LOSTA RB</i>	arbitration lost in master mode interrupt flag
<i>I2C_INT_FLAG_AERR</i>	acknowledge error interrupt flag
<i>I2C_INT_FLAG_OUER R</i>	over-run or under-run situation occurs in slave mode interrupt flag
<i>I2C_INT_FLAG_PCE RR</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBT O</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBA LT</i>	SMBus Alert status interrupt flag
<i>I2C_INT_FLAG_TFF</i>	txframe fall interrupt flag
<i>I2C_INT_FLAG_TFR</i>	txframe rise interrupt flag
<i>I2C_INT_FLAG_RFF</i>	rxframe fall interrupt flag
<i>I2C_INT_FLAG_RFR</i>	rxframe rise interrupt flag
<i>I2C_INT_FLAG_STLO</i>	start lost interrupt flag
<i>I2C_INT_FLAG_STPSE ND</i>	stop condition sent interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* check the byte transmission finishes interrupt flag is set or not*/
```

```
FlagStatus flag_state = RESET;
flag_state = i2c_interrupt_flag_get (I2C0, I2C_INT_FLAG_BTC);
```

i2c_interrupt_flag_clear

The description of i2c_interrupt_flag_clear is shown as below:

Table 3-624. 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
I2Cx	(x=0,1)
Input parameter{in}	
int_flag	interrupt flag, refer to Table 3-576. i2c_interrupt_flag_enum .
I2C_INT_FLAG_ADDS END	address is sent in master mode or received and matches in slave mode interrupt flag
I2C_INT_FLAG_BERR	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
I2C_INT_FLAG_LOSTA RB	arbitration lost in master mode interrupt flag
I2C_INT_FLAG_AERR	acknowledge error interrupt flag
I2C_INT_FLAG_OUER R	over-run or under-run situation occurs in slave mode interrupt flag
I2C_INT_FLAG_PECE RR	PEC error when receiving data interrupt flag
I2C_INT_FLAG_SMBT O	timeout signal in SMBus mode interrupt flag
I2C_INT_FLAG_SMBA LT	SMBus Alert status interrupt flag
I2C_INT_FLAG_TFF	txframe fall interrupt flag
I2C_INT_FLAG_TFR	txframe rise interrupt flag
I2C_INT_FLAG_RFF	rxframe fall interrupt flag
I2C_INT_FLAG_RFR	rxframe rise interrupt flag
I2C_INT_FLAG_STLO	start lost interrupt flag
I2C_INT_FLAG_STPSE ND	stop condition sent interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the acknowledge error interrupt */
i2c_interrupt_flag_clear (I2C0, I2C_INT_FLAG_AERR);
```

i2c_timing_config

The description of i2c_timing_config is shown as below:

Table 3-625. Function i2c_timing_config

Function name	i2c_timing_config
Function prototype	void i2c_timing_config(uint32_t i2c_periph, uint32_t psc, uint32_t scl_dely, uint32_t sda_dely);
Function descriptions	configure the timing parameters
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Input parameter{in}	
psc	0-0xf, timing prescaler
Input parameter{in}	
scl_dely	0-0xf,data setup time
Input parameter{in}	
sda_dely	0-0xf,data hold time
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the timing parameters */
i2c_timing_config (I2C2, 0x1, 0x2, 0x1);
```

i2c_digital_noise_filter_config

The description of i2c_digital_noise_filter_config is shown as below:

Table 3-626. Function i2c_digital_noise_filter_config

Function name	i2c_digital_noise_filter_config
Function prototype	void i2c_digital_noise_filter_config(uint32_t i2c_periph, uint32_t filter_length);
Function descriptions	configure digital noise filter
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Input parameter{in}	

filter_length	filter_length
<i>FILTER_DISABLE</i>	digital filter is disabled
<i>FILTER_LENGTH_1</i>	digital filter is enabled and filter spikes with a length of up to 1 tl2CCLK
<i>FILTER_LENGTH_2</i>	digital filter is enabled and filter spikes with a length of up to 2 tl2CCLK
<i>FILTER_LENGTH_3</i>	digital filter is enabled and filter spikes with a length of up to 3 tl2CCLK
<i>FILTER_LENGTH_4</i>	digital filter is enabled and filter spikes with a length of up to 4 tl2CCLK
<i>FILTER_LENGTH_5</i>	digital filter is enabled and filter spikes with a length of up to 5 tl2CCLK
<i>FILTER_LENGTH_6</i>	digital filter is enabled and filter spikes with a length of up to 6 tl2CCLK
<i>FILTER_LENGTH_7</i>	digital filter is enabled and filter spikes with a length of up to 7 tl2CCLK
<i>FILTER_LENGTH_8</i>	digital filter is enabled and filter spikes with a length of up to 8 tl2CCLK
<i>FILTER_LENGTH_9</i>	digital filter is enabled and filter spikes with a length of up to 9 tl2CCLK
<i>FILTER_LENGTH_10</i>	digital filter is enabled and filter spikes with a length of up to 10 tl2CCLK
<i>FILTER_LENGTH_11</i>	digital filter is enabled and filter spikes with a length of up to 11 tl2CCLK
<i>FILTER_LENGTH_12</i>	digital filter is enabled and filter spikes with a length of up to 12 tl2CCLK
<i>FILTER_LENGTH_13</i>	digital filter is enabled and filter spikes with a length of up to 13 tl2CCLK
<i>FILTER_LENGTH_14</i>	digital filter is enabled and filter spikes with a length of up to 14 tl2CCLK
<i>FILTER_LENGTH_15</i>	digital filter is enabled and filter spikes with a length of up to 15 tl2CCLK
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C2 digital filter filters spikes with a length of up to 1 tl2CCLK */
i2c_digital_noise_filter_config (I2C2, FILTER_LENGTH_1);
```

i2c_analog_noise_filter_enable

The description of i2c_analog_noise_filter_enable is shown as below:

Table 3-627. Function i2c_analog_noise_filter_enable

Function name	i2c_analog_noise_filter_enable
Function prototype	void i2c_analog_noise_filter_enable(uint32_t i2c_periph);
Function descriptions	enable analog noise filter
Precondition	-
The called functions	-
Input parameter{in}	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable analog noise filter */

i2c_analog_noise_filter_enable (I2C2);
```

i2c_analog_noise_filter_disable

The description of i2c_analog_noise_filter_disable is shown as below:

Table 3-628. Function i2c_analog_noise_filter_disable

Function name	i2c_analog_noise_filter_disable
Function prototype	void i2c_analog_noise_filter_disable(uint32_t i2c_periph);
Function descriptions	disable analog noise filter
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable analog noise filter */

i2c_analog_noise_filter_disable (I2C2);
```

i2c_wakeup_from_deepsleep_enable

The description of i2c_wakeup_from_deepsleep_enable is shown as below:

Table 3-629. Function i2c_wakeup_from_deepsleep_enable

Function name	i2c_wakeup_from_deepsleep_enable
Function prototype	void i2c_wakeup_from_deepsleep_enable(uint32_t i2c_periph);
Function descriptions	enable wakeup from Deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable wakeup from Deep-sleep mode */

i2c_wakeup_from_deepsleep_enable (I2C2);
```

i2c_wakeup_from_deepsleep_disable

The description of i2c_wakeup_from_deepsleep_disable is shown as below:

Table 3-630. Function i2c_wakeup_from_deepsleep_disable

Function name	i2c_wakeup_from_deepsleep_disable
Function prototype	void i2c_wakeup_from_deepsleep_disable(uint32_t i2c_periph);
Function descriptions	disable wakeup from Deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable wakeup from Deep-sleep mode */

i2c_wakeup_from_deepsleep_disable (I2C2);
```

i2c_master_clock_config

The description of i2c_master_clock_config is shown as below:

Table 3-631. Function i2c_master_clock_config

Function name	i2c_master_clock_config
Function prototype	void i2c_master_clock_config(uint32_t i2c_periph, uint32_t sclh, uint32_t scll);
Function descriptions	configure the SCL high and low period of clock in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Input parameter{in}	
sclh	0-0xff, SCL high period
Input parameter{in}	

scll	0-0xff, SCL low period
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the SCL and SDA period of clock in master mode */
i2c_master_clock_config (I2C2, 0x0f, 0x0f);
```

i2c2_master_addressing

The description of i2c2_master_addressing is shown as below:

Table 3-632. Function i2c2_master_transfer_direction_config

Function name	i2c2_master_addressing
Function prototype	void i2c2_master_addressing(uint32_t i2c_periph, uint32_t address, uint32_t trans_direction);
Function descriptions	configure i2c slave addresss and transfer direction in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Input parameter{in}	
address	0-0x3FF except reserved address, I2C slave address to be sent
Input parameter{in}	
trans_direction	I2C transfer direction in master mode
I2C2_MASTER_TRANSMIT	master transmit
I2C2_MASTER_RECEIVE	master receive
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* send slave address to I2C bus */
i2c2_master_addressing (I2C2, 0x82, I2C2_MASTER_TRANSMIT);
```

i2c_address10_header_enable

The description of i2c_address10_header_enable is shown as below:

Table 3-633. Function i2c_address10_header_enable

Function name	i2c_address10_header_enable
Function prototype	void i2c_address10_header_enable(uint32_t i2c_periph);
Function descriptions	10-bit address header executes read direction only in master receive mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 10-bit address header executes read direction only in master receive mode */

i2c_address10_header_enable (I2C2);
```

i2c_address10_header_disable

The description of i2c_address10_header_disable is shown as below:

Table 3-634. Function i2c_address10_header_disable

Function name	i2c_address10_header_disable
Function prototype	void i2c_address10_header_disable(uint32_t i2c_periph);
Function descriptions	10-bit address header executes complete sequence in master receive mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 10-bit address header executes complete sequence in master receive mode */

i2c_address10_header_disable (I2C2);
```

i2c_address10_enable

The description of i2c_address10_enable is shown as below:

Table 3-635. Function i2c_address10_enable

Function name	i2c_address10_enable
Function prototype	void i2c_address10_enable(uint32_t i2c_periph);
Function descriptions	enable 10-bit addressing mode in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable 10-bit addressing mode in master mode */

i2c_address10_enable (I2C2);
```

i2c_address10_disable

The description of i2c_address10_disable is shown as below:

Table 3-636. Function i2c_address10_disable

Function name	i2c_address10_disable
Function prototype	void i2c_address10_disable(uint32_t i2c_periph);
Function descriptions	disable 10-bit addressing mode in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable 10-bit addressing mode in master mode */

i2c_address10_disable (I2C2);
```

i2c_automatic_end_enable

The description of i2c_automatic_end_enable is shown as below:

Table 3-637. Function i2c_automatic_end_enable

Function name	i2c_automatic_end_enable
Function prototype	void i2c_automatic_end_enable(uint32_t i2c_periph);
Function descriptions	enable I2C automatic end mode in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C automatic end mode in master mode */

i2c_automatic_end_enable (I2C2);
```

i2c_automatic_end_disable

The description of i2c_automatic_end_disable is shown as below:

Table 3-638. Function i2c_automatic_end_disable

Function name	i2c_automatic_end_disable
Function prototype	void i2c_automatic_end_disable(uint32_t i2c_periph);
Function descriptions	disable I2C automatic end mode in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C automatic end mode in master mode */

i2c_automatic_end_disable (I2C2);
```

i2c_address_config

The description of i2c_address_config is shown as below:

Table 3-639. Function i2c_address_config

Function name	i2c_address_config
Function prototype	void i2c_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_format);
Function descriptions	configure i2c slave address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Input parameter{in}	
address	I2C address
Input parameter{in}	
addr_format	7bits or 10bits
I2C_ADDFORMAT_7BITS	7bits
I2C_ADDFORMAT_10BITS	10bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure i2c slave address */

i2c_address_config (I2C2, 0x82, I2C_ADDFORMAT_7BITS);
```

i2c_address_disable

The description of i2c_address_disable is shown as below:

Table 3-640. Function i2c_address_disable

Function name	i2c_address_disable
Function prototype	void i2c_address_disable(uint32_t i2c_periph);
Function descriptions	disable i2c address in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable i2c address in slave mode */

i2c_address_disable (I2C2);
```

i2c_second_address_config

The description of i2c_second_address_config is shown as below:

Table 3-641. Function i2c_second_address_config

Function name	i2c_second_address_config
Function prototype	void i2c_second_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_mask);
Function descriptions	configure i2c second slave address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Input parameter{in}	
address	I2C address
Input parameter{in}	
addr_mask	the bits not need to compare
ADDRESS2_NO_MASK_K	no mask, all the bits must be compared
ADDRESS2_MASK_BI_T1	ADDRESS2[1] is masked, only ADDRESS2[7:2] are compared
ADDRESS2_MASK_BI_T1_2	ADDRESS2[2:1] is masked, only ADDRESS2[7:3] are compared
ADDRESS2_MASK_BI_T1_3	ADDRESS2[3:1] is masked, only ADDRESS2[7:4] are compared
ADDRESS2_MASK_BI_T1_4	ADDRESS2[4:1] is masked, only ADDRESS2[7:5] are compared
ADDRESS2_MASK_BI_T1_5	ADDRESS2[5:1] is masked, only ADDRESS2[7:6] are compared
ADDRESS2_MASK_BI_T1_6	ADDRESS2[6:1] is masked, only ADDRESS2[7] are compared
ADDRESS2_MASK_ALL	all the ADDRESS2[7:1] bits are masked

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure i2c second slave address */

i2c_second_address_config (I2C2, 0x82, ADDRESS2_MASK_BIT1_2);
```

i2c_second_address_disable

The description of i2c_second_address_disable is shown as below:

Table 3-642. Function i2c_second_address_disable

Function name	i2c_second_address_disable
Function prototype	void i2c_second_address_disable(uint32_t i2c_periph);
Function descriptions	disable i2c second address in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable i2c second address in slave mode */

i2c_second_address_disable (I2C2);
```

i2c_recevied_address_get

The description of i2c_recevied_address_get is shown as below:

Table 3-643. Function i2c_recevied_address_get

Function name	i2c_recevied_address_get
Function prototype	uint32_t i2c_recevied_address_get(uint32_t i2c_periph);
Function descriptions	get received match address in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)

Output parameter{out}	
-	-
Return value	
uint32_t	0x00..0x7F

Example:

```
/* get received match address in slave mode */

uint32_t address;

address = i2c_recevied_address_get (I2C2);
```

i2c_slave_byte_control_enable

The description of i2c_slave_byte_control_enable is shown as below:

Table 3-644. Function i2c_slave_byte_control_enable

Function name	i2c_slave_byte_control_enable
Function prototype	void i2c_slave_byte_control_enable(uint32_t i2c_periph);
Function descriptions	enable slave byte control
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable slave byte control */

i2c_slave_byte_control_enable (I2C2);
```

i2c_slave_byte_control_disable

The description of i2c_slave_byte_control_disable is shown as below:

Table 3-645. Function i2c_slave_byte_control_disable

Function name	i2c_slave_byte_control_disable
Function prototype	void i2c_slave_byte_control_disable(uint32_t i2c_periph);
Function descriptions	disable slave byte control
Precondition	-
The called functions	-
Input parameter{in}	

i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable slave byte control */

i2c_slave_byte_control_disable (I2C2);
```

i2c_nack_enable

The description of **i2c_nack_enable** is shown as below:

Table 3-646. Function i2c_nack_enable

Function name	i2c_nack_enable
Function prototype	void i2c_nack_enable(uint32_t i2c_periph);
Function descriptions	generate a NACK in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* generate a NACK in slave mode */

i2c_nack_enable (I2C2);
```

i2c_nack_disable

The description of **i2c_nack_disable** is shown as below:

Table 3-647. Function i2c_nack_disable

Function name	i2c_nack_disable
Function prototype	void i2c_nack_disable(uint32_t i2c_periph);
Function descriptions	generate a ACK in slave mode
Precondition	-
The called functions	-
Input parameter{in}	

i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* generate a ACK in slave mode */

i2c_nack_disable (I2C2);
```

i2c_reload_enable

The description of **i2c_reload_enable** is shown as below:

Table 3-648. Function i2c_reload_enable

Function name	i2c_reload_enable
Function prototype	void i2c_reload_enable(uint32_t i2c_periph);
Function descriptions	enable I2C reload mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C reload mode */

i2c_reload_enable (I2C2);
```

i2c_reload_disable

The description of **i2c_reload_disable** is shown as below:

Table 3-649. Function i2c_reload_disable

Function name	i2c_reload_disable
Function prototype	void i2c_reload_disable(uint32_t i2c_periph);
Function descriptions	disable I2C reload mode
Precondition	-
The called functions	-
Input parameter{in}	

i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C reload mode */

i2c_reload_disable (I2C2);
```

i2c_transfer_byte_number_config

The description of `i2c_transfer_byte_number_config` is shown as below:

Table 3-650. Function `i2c_transfer_byte_number_config`

Function name	<code>i2c_transfer_byte_number_config</code>
Function prototype	<code>void i2c_transfer_byte_number_config(uint32_t i2c_periph, uint32_t byte_number);</code>
Function descriptions	configure number of bytes to be transferred
Precondition	-
The called functions	-
Input parameter{in}	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=2)
Input parameter{in}	
byte_number	0x0-0xFF, number of bytes to be transferred
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure number of bytes to be transferred */

i2c_transfer_byte_number_config (I2C2, 0xFF);
```

i2c2_dma_enable

The description of `i2c2_dma_enable` is shown as below:

Table 3-651. Function `i2c2_dma_enable`

Function name	<code>i2c2_dma_enable</code>
Function prototype	<code>void i2c2_dma_enable(uint32_t i2c_periph, uint8_t dma);</code>
Function descriptions	enable I2C DMA for transmission or reception

Precondition	-
The called functions	-
Input parameter{in}	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=2)
Input parameter{in}	
dma	I2C DMA
<i>I2C2_DMA_TRANSMIT</i>	transmit data using DMA
<i>I2C2_DMA_RECEIVE</i>	receive data using DMA
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C DMA for transmission or reception */

i2c2_dma_enable (I2C2, I2C2_DMA_RECEIVE);
```

i2c2_dma_disable

The description of `i2c2_dma_disable` is shown as below:

Table 3-652. Function i2c2_dma_disable

Function name	i2c2_dma_disable
Function prototype	void i2c2_dma_disable(uint32_t i2c_periph, uint8_t dma);
Function descriptions	disable I2C DMA for transmission or reception
Precondition	-
The called functions	-
Input parameter{in}	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=2)
Input parameter{in}	
dma	I2C DMA
<i>I2C2_DMA_TRANSMIT</i>	transmit data using DMA
<i>I2C2_DMA_RECEIVE</i>	receive data using DMA
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C DMA for transmission or reception */

i2c2_dma_disable (I2C2, I2C2_DMA_RECEIVE);
```

i2c_smbus_alert_enable

The description of i2c_smbus_alert_enable is shown as below:

Table 3-653. Function i2c_smbus_alert_enable

Function name	i2c_smbus_alert_enable
Function prototype	void i2c_smbus_alert_enable(uint32_t i2c_periph);
Function descriptions	enable SMBus Alert
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SMBus Alert */

i2c_smbus_alert_enable (I2C2);
```

i2c_smbus_alert_disable

The description of i2c_smbus_alert_disable is shown as below:

Table 3-654. Function i2c_smbus_alert_disable

Function name	i2c_smbus_alert_disable
Function prototype	void i2c_smbus_alert_disable(uint32_t i2c_periph);
Function descriptions	disable SMBus Alert
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SMBus Alert */

i2c_smbus_alert_disable (I2C2);
```

i2c_smbus_default_addr_enable

The description of i2c_smbus_default_addr_enable is shown as below:

Table 3-655. Function i2c_smbus_default_addr_enable

Function name	i2c_smbus_default_addr_enable
Function prototype	void i2c_smbus_default_addr_enable(uint32_t i2c_periph);
Function descriptions	enable SMBus device default address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SMBus device default address */

i2c_smbus_default_addr_enable (I2C2);
```

i2c_smbus_default_addr_disable

The description of i2c_smbus_default_addr_disable is shown as below:

Table 3-656. Function i2c_smbus_default_addr_disable

Function name	i2c_smbus_default_addr_disable
Function prototype	void i2c_smbus_default_addr_disable(uint32_t i2c_periph);
Function descriptions	disable SMBus device default address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SMBus device default address */

i2c_smbus_default_addr_disable (I2C2);
```

i2c_smbus_host_addr_enable

The description of i2c_smbus_host_addr_enable is shown as below:

Table 3-657. Function i2c_smbus_host_addr_enable

Function name	i2c_smbus_host_addr_enable
Function prototype	void i2c_smbus_host_addr_enable(uint32_t i2c_periph);
Function descriptions	enable SMBus Host address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SMBus Host address */
i2c_smbus_host_addr_enable (I2C2);
```

i2c_smbus_host_addr_disable

The description of i2c_smbus_host_addr_disable is shown as below:

Table 3-658. Function i2c_smbus_host_addr_disable

Function name	i2c_smbus_host_addr_disable
Function prototype	void i2c_smbus_host_addr_disable(uint32_t i2c_periph);
Function descriptions	disable SMBus Host address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SMBus Host address */
i2c_smbus_host_addr_disable (I2C2);
```

i2c_extented_clock_timeout_enable

The description of i2c_extented_clock_timeout_enable is shown as below:

Table 3-659. Function i2c_extented_clock_timeout_enable

Function name	i2c_extented_clock_timeout_enable
Function prototype	void i2c_extented_clock_timeout_enable(uint32_t i2c_periph);
Function descriptions	enable extended clock timeout detection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable extended clock timeout detection */
i2c_extented_clock_timeout_enable (I2C2);
```

i2c_extented_clock_timeout_disable

The description of i2c_extented_clock_timeout_disable is shown as below:

Table 3-660. Function i2c_extented_clock_timeout_disable

Function name	i2c_extented_clock_timeout_disable
Function prototype	void i2c_extented_clock_timeout_disable(uint32_t i2c_periph);
Function descriptions	disable extended clock timeout detection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable extended clock timeout detection */
i2c_extented_clock_timeout_disable (I2C2);
```

i2c_clock_timeout_enable

The description of i2c_clock_timeout_enable is shown as below:

Table 3-661. Function i2c_clock_timeout_enable

Function name	i2c_clock_timeout_enable
Function prototype	void i2c_clock_timeout_enable(uint32_t i2c_periph);
Function descriptions	enable clock timeout detection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable clock timeout detection */
i2c_clock_timeout_enable (I2C2);
```

i2c_clock_timeout_disable

The description of i2c_clock_timeout_disable is shown as below:

Table 3-662. Function i2c_clock_timeout_disable

Function name	i2c_clock_timeout_disable
Function prototype	void i2c_clock_timeout_disable(uint32_t i2c_periph);
Function descriptions	disable clock timeout detection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable clock timeout detection */
i2c_clock_timeout_disable (I2C2);
```

i2c_bus_timeout_b_config

The description of i2c_bus_timeout_b_config is shown as below:

Table 3-663. Function i2c_bus_timeout_b_config

Function name	i2c_bus_timeout_b_config
Function prototype	void i2c_bus_timeout_b_config(uint32_t i2c_periph, uint32_t timeout);
Function descriptions	configure bus timeout B
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Input parameter{in}	
timeout	0-0xffff, bus timeout B
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure bus timeout B */
i2c_bus_timeout_b_config (I2C2, 0xff);
```

i2c_bus_timeout_a_config

The description of i2c_bus_timeout_a_config is shown as below:

Table 3-664. Function i2c_bus_timeout_a_config

Function name	i2c_bus_timeout_a_config
Function prototype	void i2c_bus_timeout_a_config(uint32_t i2c_periph, uint32_t timeout);
Function descriptions	configure bus timeout A
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Input parameter{in}	
timeout	0-0xffff, bus timeout A
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure bus timeout A */

i2c_bus_timeout_a_config (I2C2, 0xff);
```

i2c_idle_clock_timeout_config

The description of i2c_idle_clock_timeout_config is shown as below:

Table 3-665. Function i2c_idle_clock_timeout_config

Function name	i2c_idle_clock_timeout_config
Function prototype	void i2c_idle_clock_timeout_config(uint32_t i2c_periph, uint32_t timeout);
Function descriptions	configure idle clock timeout detection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Input parameter{in}	
timeout	bus timeout A
BUSTOA_DETECT_SC_L_LOW	BUSTOA is used to detect SCL low timeout
BUSTOA_DETECT_IDLE	BUSTOA is used to detect both SCL and SDA high timeout when the bus is idle
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure idle clock timeout detection */

i2c_idle_clock_timeout_config (I2C2, BUSTOA_DETECT_SCL_LOW);
```

i2c2_flag_get

The description of i2c2_flag_get is shown as below:

Table 3-666. Function i2c2_flag_get

Function name	i2c2_flag_get
Function prototype	FlagStatus i2c2_flag_get(uint32_t i2c_periph, uint32_t flag);
Function descriptions	get I2C flag status
Precondition	-
The called functions	-
Input parameter{in}	

i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=2)
Input parameter{in}	
flag	I2C flags
<i>I2C2_FLAG_TBE</i>	I2C2_TDATA is empty during transmitting
<i>I2C2_FLAG_TI</i>	transmit interrupt
<i>I2C2_FLAG_RBNE</i>	I2C2_RDATA is not empty during receiving
<i>I2C2_FLAG_ADDSEND</i>	address received matches in slave mode
<i>I2C2_FLAG_NACK</i>	not acknowledge flag
<i>I2C2_FLAG_STPDET</i>	STOP condition detected in slave mode
<i>I2C2_FLAG_TC</i>	transfer complete in master mode
<i>I2C2_FLAG_TCR</i>	transfer complete reload
<i>I2C2_FLAG_BERR</i>	bus error
<i>I2C2_FLAG_LOSTARB</i>	arbitration Lost
<i>I2C2_FLAG_OUERR</i>	overrun/underrun error in slave mode
<i>I2C2_FLAG_PECERR</i>	PEC error
<i>I2C2_FLAG_TIMEOUT</i>	timeout flag
<i>I2C2_FLAG_SMBALT</i>	SMBus Alert
<i>I2C2_FLAG_I2CBSY</i>	busy flag
<i>I2C2_FLAG_TR</i>	whether the I2C is a transmitter or a receiver in slave mode
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* get I2C flag status */

FlagStatus flag_state = RESET;

flag_state = i2c2_flag_get (I2C2, I2C2_FLAG_TBE);
```

i2c2_flag_clear

The description of i2c2_flag_clear is shown as below:

Table 3-667. Function i2c2_flag_clear

Function name	i2c2_flag_clear
Function prototype	void i2c2_flag_clear(uint32_t i2c_periph, uint32_t flag);
Function descriptions	clear I2C flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=2)

Input parameter{in}	
flag	I2C flags
<i>I2C2_FLAG_ADDSEND</i>	address received matches in slave mode
<i>I2C2_FLAG_NACK</i>	not acknowledge flag
<i>I2C2_FLAG_STPDET</i>	STOP condition detected in slave mode
<i>I2C2_FLAG_BERR</i>	bus error
<i>I2C2_FLAG_LOSTARB</i>	arbitration Lost
<i>I2C2_FLAG_OUERR</i>	overrun/underrun error in slave mode
<i>I2C2_FLAG_PECERR</i>	PEC error
<i>I2C2_FLAG_TIMEOUT</i>	timeout flag
<i>I2C2_FLAG_SMBALT</i>	SMBus Alert
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear a bus error flag*/
i2c2_flag_clear (I2C2, I2C2_FLAG_BERR);
```

i2c2_interrupt_enable

The description of i2c2_interrupt_enable is shown as below:

Table 3-668. Function i2c2_interrupt_enable

Function name	i2c2_interrupt_enable
Function prototype	void i2c2_interrupt_enable(uint32_t i2c_periph, uint32_t interrupt);
Function descriptions	enable I2C interrupt
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=2)
Input parameter{in}	
interrupt	I2C interrupts
<i>I2C2_INT_ERR</i>	error interrupt
<i>I2C2_INT_TC</i>	transfer complete interrupt
<i>I2C2_INT_STPDET</i>	stop detection interrupt
<i>I2C2_INT_NACK</i>	not acknowledge received interrupt
<i>I2C2_INT_ADDM</i>	address match interrupt
<i>I2C2_INT_RBNE</i>	receive interrupt
<i>I2C2_INT_TI</i>	transmit interrupt
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable I2C2 transmit interrupt */
i2c2_interrupt_enable (I2C2, I2C2_INT_TI);
```

i2c2_interrupt_disable

The description of i2c2_interrupt_disable is shown as below:

Table 3-669. Function i2c2_interrupt_disable

Function name	i2c2_interrupt_disable
Function prototype	void i2c2_interrupt_disable(uint32_t i2c_periph, uint32_t interrupt);
Function descriptions	disable I2C interrupt
Precondition	-
The called functions	-
Input parameter{in}	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=2)
Input parameter{in}	
<i>interrupt</i>	I2C interrupts
<i>I2C2_INT_ERR</i>	error interrupt
<i>I2C2_INT_TC</i>	transfer complete interrupt
<i>I2C2_INT_STPDET</i>	stop detection interrupt
<i>I2C2_INT_NACK</i>	not acknowledge received interrupt
<i>I2C2_INT_ADDM</i>	address match interrupt
<i>I2C2_INT_RBNE</i>	receive interrupt
<i>I2C2_INT_TI</i>	transmit interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C2 transmit interrupt */
i2c2_interrupt_disable (I2C2, I2C2_INT_TI);
```

i2c2_interrupt_flag_get

The description of i2c2_interrupt_flag_get is shown as below:

Table 3-670. Function i2c2_interrupt_flag_get

Function name	i2c2_interrupt_flag_get
Function prototype	FlagStatus i2c2_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
Function descriptions	get I2C interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Input parameter{in}	
int_flag	I2C interrupt flags, refer to Table 3-577. i2c2_interrupt_flag_enum .
I2C2_INT_FLAG_TI	transmit interrupt flag
I2C2_INT_FLAG_RBN_E	I2C2_RDATA is not empty during receiving interrupt flag
I2C2_INT_FLAG_ADD_SEND	address received matches in slave mode interrupt flag
I2C2_INT_FLAG_NACK_K	not acknowledge interrupt flag
I2C2_INT_FLAG_STPD_ET	stop condition detected in slave mode interrupt flag
I2C2_INT_FLAG_TC	transfer complete in master mode interrupt flag
I2C2_INT_FLAG_TCR	transfer complete reload interrupt flag
I2C2_INT_FLAG_BER_R	bus error interrupt flag
I2C2_INT_FLAG_LOST_ARB	arbitration lost interrupt flag
I2C2_INT_FLAG_OUE_RR	overrun/underrun error in slave mode interrupt flag
I2C2_INT_FLAG_PECERR_RR	PEC error interrupt flag
I2C2_INT_FLAG_TIME_OUT	timeout interrupt flag
I2C2_INT_FLAG_SMB_ALERT	SMBus Alert interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* get I2C interrupt flag status */
```

```

FlagStatus flag_state = RESET;
flag_state = i2c2_interrupt_flag_get (I2C2, I2C2_INT_FLAG_TI);
  
```

i2c2_interrupt_flag_clear

The description of i2c2_interrupt_flag_clear is shown as below:

Table 3-671. Function i2c2_interrupt_flag_clear

Function name	i2c2_interrupt_flag_clear
Function prototype	void i2c2_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
Function descriptions	clear I2C interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=2)
Input parameter{in}	
int_flag	I2C interrupt flags, refer to Table 3-577. i2c2_interrupt_flag_enum .
I2C2_INT_FLAG_ADD SEND	address received matches in slave mode interrupt flag
I2C2_INT_FLAG_NAC K	not acknowledge interrupt flag
I2C2_INT_FLAG_STPD ET	stop condition detected in slave mode interrupt flag
I2C2_INT_FLAG_BER R	bus error interrupt flag
I2C2_INT_FLAG_LOST ARB	arbitration lost interrupt flag
I2C2_INT_FLAG_OUE RR	overrun/underrun error in slave mode interrupt flag
I2C2_INT_FLAG_PEC RR	PEC error interrupt flag
I2C2_INT_FLAG_TIME OUT	timeout interrupt flag
I2C2_INT_FLAG_SMB ALT	SMBus Alert interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* clear a bus error flag */

i2c2_interrupt_flag_clear (I2C2, I2C2_INT_FLAG_BERR);
  
```

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 3-672. NVIC Registers

Registers	Descriptions
ISER ⁽¹⁾	Interrupt Set Enable Register
ICER ⁽¹⁾	Interrupt Clear Enable Register
ISPR ⁽¹⁾	Interrupt Set Pending Register
ICPR ⁽¹⁾	Interrupt Clear Pending Register
IABR ⁽¹⁾	Interrupt Active bit Register
ITNS ⁽¹⁾	Interrupt Non-Secure State Register
IPR ⁽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
SHPR ⁽²⁾	System Handlers Priority Registers
SHCSR ⁽²⁾	System Handler Control and State Register

1. refer to the structure NVIC_Type, is defined in the core_cm33.h file

2. refer to the structure SCB_Type, is defined in the core_cm33.h file

Table 3-673. 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_cm33.h file

3.19.2. Descriptions of Peripheral functions

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

Table 3-674. MISC firmware function

Function name	Function description
nvic_priority_group_set	set the priority group
nvic_irq_enable	enable NVIC interrupt request
nvic_irq_disable	disable NVIC interrupt request
nvic_system_reset	initiates a system reset request to reset the MCU
nvic_vector_table_set	set the NVIC vector table base address
system_lowpower_set	set the state of the low power mode
system_lowpower_reset	reset the state of the low power mode
systick_clksource_set	set the systick clock source

Enum IRQn_Type

Table 3-675. IRQn_Type

Member name	Function description
WWDGT_IRQHandler	window watchDog timer interrupt
LVD_IRQHandler	LVD through EXTI line detect interrupt
TAMPER_IRQHandler	tamper through EXTI line detect
RTC_IRQHandler	RTC through EXTI line interrupt
FMC_IRQHandler	FMC interrupt
RCU_CTC_IRQHandler	RCU and CTC interrupt
EXTI0_IRQHandler	EXTI line 0 interrupts
EXTI1_IRQHandler	EXTI line 1 interrupts
EXTI2_IRQHandler	EXTI line 2 interrupts
EXTI3_IRQHandler	EXTI line 3 interrupts
EXTI4_IRQHandler	EXTI line 4 interrupts
DMA0_Channel0_IRQHandler	DMA0 channel0 interrupt
DMA0_Channel1_IRQHandler	DMA0 channel1 interrupt
DMA0_Channel2_IRQHandler	DMA0 channel2 interrupt
DMA0_Channel3_IRQHandler	DMA0 channel3 interrupt
DMA0_Channel4_IRQHandler	DMA0 channel4 interrupt
DMA0_Channel5_IRQHandler	DMA0 channel5 interrupt
DMA0_Channel6_IRQHandler	DMA0 channel6 interrupt
ADC0_1_IRQHandler	ADC0 and ADC1 interrupt
USBD_HP_CAN0_TX_IRQHandler / CAN0_TX_IRQHandler	USBD High Priority or CAN0 TX interrupts / CAN0 transmit interrupt
USBD_LP_CAN0_RX0_IRQHandler / CAN0_RX0_IRQHandler	USBD Low Priority or CAN0 RX0 interrupts / CAN0 receive0 interrupts
CAN0_RX1_IRQHandler	CAN0 receive1 interrupts
CAN0_EWMC_IRQHandler	CAN0 EWMC interrupts
EXTI5_9_IRQHandler	EXTI[9:5] interrupts
TIMER0_BRK_TIMER8_IRQHandler	TIMER0 break and TIMER8 interrupts
TIMER0_UP_TIMER9_IRQHandler	TIMER0 update and TIMER9 interrupts

Member name	Function description
TIMER0_TRG_CMT_TIMER10_IRQn	TIMER0 trigger and commutation and TIMER10 interrupts
TIMER0_Channel_IRQn	TIMER0 channel capture compare interrupt
TIMER1_IRQn	TIMER1 interrupt
TIMER2_IRQn	TIMER2 interrupt
TIMER3_IRQn	TIMER3 interrupt
I2C0_EV_IRQn	I2C0 event interrupt
I2C0_ER_IRQn	I2C0 error interrupt
I2C1_EV_IRQn	I2C1 event interrupt
I2C1_ER_IRQn	I2C1 error interrupt
SPI0_IRQn	SPI0 interrupt
SPI1_I2S1ADD_IRQn	SPI1 or I2S1ADD interrupt
USART0_IRQn	USART0 interrupt
USART1_IRQn	USART1 interrupt
USART2_IRQn	USART2 interrupt
EXTI10_15_IRQn	EXTI[15:10] interrupts
RTC_Alarm_IRQn	RTC alarm interrupt
USBD_WKUP_IRQn / USBHS_WKUP_IRQn	USBD / USBHS wakeup interrupt
TIMER7_BRK_TIMER11_IRQn	TIMER7 break and TIMER11 interrupts
TIMER7_UP_TIMER12_IRQn	TIMER7 update and TIMER12 interrupts
TIMER7_TRG_CMT_TIMER13_IRQn	TIMER7 trigger and commutation and TIMER13 interrupts
TIMER7_Channel_IRQn	TIMER7 channel capture compare interrupt
EXMC_IRQn	EXMC global interrupt
SDIO_IRQn	SDIO global interrupt
TIMER4_IRQn	TIMER4 global interrupt
SPI2_I2S2ADD_IRQn	SPI2 or I2S2ADD global interrupt
UART3_IRQn	UART3 global interrupt
UART4_IRQn	UART4 global interrupt
TIMER5_DAC_IRQn	TIMER5 or DAC global interrupt
TIMER6_IRQn	TIMER6 global interrupt
DMA1_Channel0_IRQn	DMA1 channel0 global interrupt
DMA1_Channel1_IRQn	DMA1 channel1 global interrupt
DMA1_Channel2_IRQn	DMA1 channel2 global interrupt
DMA1_Channel3_Channel4_IRQn / DMA1_Channel3_IRQn	DMA1 channel3 and channel4 global interrupt / DMA1 channel3 global interrupt
DMA1_Channel4_IRQn	DMA1 channel4 global interrupt
ENET_IRQn	ENET global interrupt
ENET_WKUP_IRQn	ENET wakeup interrupt
CAN1_TX_IRQn	CAN1 transmit interrupt
CAN1_RX0_IRQn	CAN1 receive0 interrupt
CAN1_RX1_IRQn	CAN1 receive1 interrupt

Member name	Function description
CAN1_EWMC IRQn	CAN1 EWMC interrupt
USBHS IRQn	USBHS global interrupt
SHRTIMER IRQ2 IRQn	SHRTIMER IRQ2 interrupt
SHRTIMER IRQ3 IRQn	SHRTIMER IRQ3 interrupt
SHRTIMER IRQ4 IRQn	SHRTIMER IRQ4 interrupt
SHRTIMER IRQ5 IRQn	SHRTIMER IRQ5 interrupt
SHRTIMER IRQ6 IRQn	SHRTIMER IRQ6 interrupt
USBHS EP1 OUT IRQn	USBHS end point 1 out interrupt
USBHS EP1 IN IRQn	USBHS end point 1 in interrupt
SHRTIMER IRQ0 IRQn	SHRTIMER IRQ0 interrupt
SHRTIMER IRQ1 IRQn	SHRTIMER IRQ1 interrupt
CAN2 TX IRQn	CAN2 TX interrupt
CAN2 RX0 IRQn	CAN2 RX0 interrupt
CAN2 RX1 IRQn	CAN2 RX1 interrupt
CAN2 EWMC IRQn	CAN2 EWMC interrupt
I2C2 EV IRQn	I2C2 EV interrupt
I2C2 ER IRQn	I2C2 ER interrupt
USART5 IRQn	USART5 global interrupt
I2C2 WKUP IRQn	I2C2 Wakeup interrupt
USART5 WKUP IRQn	USART5 Wakeup interrupt
TMU IRQn	TMU interrupt

nvic_priority_group_set

The description of nvic_priority_group_set is shown as below:

Table 3-676. Function nvic_priority_group_set

Function name	nvic_priority_group_set
Function prototype	void nvic_priority_group_set(uint32_t nvic_prigroup);
Function descriptions	set the priority group
Precondition	-
The called functions	-
Input parameter{in}	
nvic_prigroup	priority group
NVIC_PRIGROUP_PR E0_SUB4	0 bits for pre-emption priority 4 bits for subpriority
NVIC_PRIGROUP_PR E1_SUB3	1 bits for pre-emption priority 3 bits for subpriority
NVIC_PRIGROUP_PR E2_SUB2	2 bits for pre-emption priority 2 bits for subpriority
NVIC_PRIGROUP_PR E3_SUB1	3 bits for pre-emption priority 1 bits for subpriority

NVIC_PRIGROUP_PR E4_SUB0	4 bits for pre-emption priority 0 bits for subpriority
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* priority group configuration, 0 bits for pre-emption priority 4 bits for subpriority */
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

nvic_irq_enable

The description of nvic_irq_enable is shown as below:

Table 3-677. 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 Table 3-675. IRQn_Type
Input parameter{in}	
nvic_irq_pre_priority	the pre-emption priority needed to set
Input parameter{in}	
nvic_irq_sub_priority	the subpriority needed to set
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 3-678. 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 Table 3-675. IRQn Type
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable window watchDog timer interrupt */
nvic_irq_disable(WWDGT_IRQn);
```

nvic_system_reset

The description of nvic_system_reset is shown as below:

Table 3-679. Function nvic_system_reset

Function name	nvic_system_reset
Function prototype	void nvic_system_reset(void);
Function descriptions	initiates a system reset request to reset the MCU
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initiates a system reset request to reset the MCU */
nvic_system_reset();
```

nvic_vector_table_set

The description of nvic_vector_table_set is shown as below:

Table 3-680. Function nvic_vector_table_set

Function name	nvic_vector_table_set
Function prototype	void nvic_vector_table_set(uint32_t nvic_vict_tab, uint32_t offset);
Function descriptions	set the NVIC vector table base address
Precondition	-
The called functions	-

Input parameter{in}	
nvic_vict_tab	the RAM or FLASH base address
NVIC_VECTTAB_RAM	RAM base address
NVIC_VECTTAB_FLASH <i>H</i>	Flash base address
Input parameter{in}	
offset	vector table offset (vector table start address= base address+offset)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set vector table address = NVIC_VECTTAB_FLASH +0x200 */
nvic_vector_table_set (NVIC_VECTTAB_FLASH,0x200);
```

system_lowpower_set

The description of system_lowpower_set is shown as below:

Table 3-681. 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 */
system_lowpower_set (SCB_LPM_SLEEP_EXIT_ISR);
```

system_lowpower_reset

The description of system_lowpower_reset is shown as below:

Table 3-682. 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 woken up by the enable interrupts
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* the system will exit low power mode by exiting from ISR */
system_lowpower_reset (SCB_LPM_SLEEP_EXIT_ISR);
```

systick_clksource_set

The description of systick_clksource_set is shown as below:

Table 3-683. 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_CLKSOURCE_HCLK	systick clock source is from HCLK
SYSTICK_CLKSOURCE_HCLK_DIV8	systick clock source is from HCLK/8
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* systick clock source is HCLK/8 */
systick_clksource_set(SYSTICK_CLKSOURCE_HCLK_DIV8);
```

3.20. PMU

According to the Power management unit (PMU), provides five types of power saving modes, including Sleep, Deep-sleep, Deep-sleep 1, Deep-sleep 2 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 3-684. PMU Registers

Registers	Descriptions
PMU_CTL0	Control register 0
PMU_CS0	Control and status register 0
PMU_CTL1	Control register 1
PMU_CS1	Control and status register 1

3.20.2. Descriptions of Peripheral functions

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

Table 3-685. PMU firmware function

Function name	Function description
pmu_deinit	deinitialize the PMU
pmu_lvd_select	select low voltage detector threshold
pmu_lvd_disable	disable PMU lvd
pmu_highdriver_mode_enable	enable high-driver mode
pmu_highdriver_mode_disable	disable high-driver mode
pmu_highdriver_switch_select	switch high-driver mode
pmu_lowdriver_mode_enable	enable low-driver mode in deep-sleep mode
pmu_lowdriver_mode_disable	disable low-driver mode in deep-sleep mode
pmu_lowpower_driver_config	in deep-sleep mode, driver mode when use low power LDO
pmu_normalpower_driver_config	in deep-sleep mode, driver mode when use normal power LDO

Function name	Function description
pmu_to_sleepmode	PMU work in sleep mode
pmu_to_deepsleepmode	PMU work in deepsleep mode
pmu_to_deepsleepmode_1	PMU work in deepsleep mode 1
pmu_to_deepsleepmode_2	PMU work in deepsleep mode 2
pmu_to_standbymode	pmu work in standby mode
pmu_wakeup_pin_enable	enable wakeup pin
pmu_wakeup_pin_disable	disable wakeup pin
pmu_backup_write_enable	enable backup domain write
pmu_backup_write_disable	disable backup domain write
pmu_flag_get	get flag state
pmu_flag_clear	clear flag bit

pmu_deinit

The description of pmu_deinit is shown as below:

Table 3-686. Function pmu_deinit

Function name	pmu_deinit
Function prototype	void pmu_deinit(void);
Function descriptions	deinitialize the PMU
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset PMU */
pmu_deinit();
```

pmu_lvd_select

The description of pmu_lvd_select is shown as below:

Table 3-687. Function pmu_lvd_select

Function name	pmu_lvd_select
Function prototype	void pmu_lvd_select(uint32_t lvdt_n);
Function descriptions	select low voltage detector threshold
Precondition	-
The called functions	-

Input parameter{in}	
<i>lvdt_n</i>	voltage threshold value
<i>PMU_LVDT_0</i>	voltage threshold is 2.1V
<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 2.6V
<i>PMU_LVDT_4</i>	voltage threshold is 2.7V
<i>PMU_LVDT_5</i>	voltage threshold is 2.9V
<i>PMU_LVDT_6</i>	voltage threshold is 3.0V
<i>PMU_LVDT_7</i>	voltage threshold is 3.1V
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select low voltage detector threshold as 3.1V */
pmu_lvd_select (PMU_LVDT_7);
```

pmu_lvd_disable

The description of pmu_lvd_disable is shown as below:

Table 3-688. 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_highdriver_mode_enable

The description of pmu_highdriver_mode_enable is shown as below:

Table 3-689. Function pmu_highdriver_mode_enable

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

Example:

```
/* enable high-driver mode */

pmu_highdriver_mode_enable();
```

pmu_highdriver_mode_disable

The description of pmu_highdriver_mode_disable is shown as below:

Table 3-690. Function pmu_highdriver_mode_disable

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

Example:

```
/* disable high-driver mode */

pmu_highdriver_mode_disable();
```

pmu_highdriver_switch_select

The description of pmu_highdriver_switch_select is shown as below:

Table 3-691. Function pmu_highdriver_switch_select

Function name	pmu_highdriver_switch_select
----------------------	------------------------------

Function prototype	void pmu_highdriver_switch_select(uint32_t highdr_switch);
Function descriptions	switch high-driver mode
Precondition	-
The called functions	pmu_flag_get()
Input parameter{in}	
highdr_switch	enable or disable high-driver mode switch
<i>PMU_HIGHDR_SWITC H_NONE</i>	disable high-driver mode switch
<i>PMU_HIGHDR_SWITC H_EN</i>	enable high-driver mode switch
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable high-driver mode switch */
pmu_highdriver_switch_select (PMU_HIGHDR_SWITCH_EN);
```

pmu_lowdriver_mode_enable

The description of pmu_lowdriver_mode_enable is shown as below:

Table 3-692. Function pmu_lowdriver_mode_enable

Function name	pmu_lowdriver_mode_enable
Function prototype	void pmu_lowdriver_mode_enable(void);
Function descriptions	enable low-driver mode in deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable low-driver mode in deep-sleep mode */
pmu_lowdriver_mode_enable ();
```

pmu_lowdriver_mode_disable

The description of pmu_lowdriver_mode_disable is shown as below:

Table 3-693. Function pmu_lowdriver_mode_disable

Function name	pmu_lowdriver_mode_disable
Function prototype	void pmu_lowdriver_mode_disable (void);
Function descriptions	enable disable low-driver mode in deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable low-driver mode in deep-sleep mode */
pmu_lowdriver_mode_disable();
```

pmu_lowpower_driver_config

The description of pmu_lowpower_driver_config is shown as below:

Table 3-694. Function pmu_lowpower_driver_config

Function name	pmu_lowpower_driver_config
Function prototype	void pmu_lowpower_driver_config(uint32_t mode);
Function descriptions	driver mode when use low power LDO
Precondition	-
The called functions	-
Input parameter{in}	
mode	driver mode
PMU_NORMALDR_LO WPWR	normal driver when use low power LDO
PMU_LOWDR_LOWPWR	low-driver mode enabled when LDEN is 11 and use low power LDO
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* normal driver when use low power LDO */
pmu_lowpower_driver_config (PMU_NORMALDR_LOWPWR);
```

pmu_normalpower_driver_config

The description of pmu_normalpower_driver_config is shown as below:

Table 3-695. Function pmu_normalpower_driver_config

Function name	pmu_normalpower_driver_config
Function prototype	void pmu_normalpower_driver_config (uint32_t mode);
Function descriptions	driver mode when use normal power LDO
Precondition	-
The called functions	-
Input parameter{in}	
mode	driver mode
<i>PMU_NORMALDR_LO WPWR</i>	normal driver when use normal power LDO
<i>PMU_LOWDR_LOWPWR WR</i>	low-driver mode enabled when LDEN is 11 and use normal power LDO
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* normal driver when use normal power LDO */
pmu_normalpower_driver_config (PMU_NORMALDR_LOWPWR);
```

pmu_to_sleepmode

The description of pmu_to_sleepmode is shown as below:

Table 3-696. Function pmu_to_sleepmode

Function name	pmu_to_sleepmode
Function prototype	void pmu_to_sleepmode(uint8_t sleepmodecmd);
Function descriptions	PMU work in 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 in sleep mode */

pmu_to_sleepmode (WFI_CMD);

```

pmu_to_deepsleepmode

The description of pmu_to_deepsleepmode is shown as below:

Table 3-697. Function pmu_to_deepsleepmode

Function name	pmu_to_deepsleepmode
Function prototype	void pmu_to_deepsleepmode(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmodecmd);
Function descriptions	PMU work in deepsleep mode
Precondition	-
The called functions	-
Input parameter{in}	
ldo	ldo work mode
<i>PMU_LDO_NORMAL</i>	LDO normal work when pmu enter deepsleep mode
<i>PMU_LDO_LOWPOW ER</i>	LDO work at low power mode when pmu enter deepsleep mode
Input parameter{in}	
lowdrive	low-driver mode
<i>PMU_LOWDRIVER_E NABLE</i>	low-driver mode enable in deep-sleep mode
<i>PMU_LOWDRIVER_D ISABLE</i>	low-driver mode disable in deep-sleep 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 in deepsleep mode */

pmu_to_deepsleepmode (PMU_LDO_NORMAL, PMU_LOWDRIVER_ENABLE, WFI_CMD);

```

pmu_to_deepsleepmode_1

The description of pmu_to_deepsleepmode_1 is shown as below:

Table 3-698. Function pmu_to_deepsleepmode_1

Function name	pmu_to_deepsleepmode_1
----------------------	------------------------

Function prototype	void pmu_to_deepsleepmode_1(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmode1cmd);
Function descriptions	PMU work in deepsleep mode 1
Precondition	-
The called functions	-
Input parameter{in}	
ldo	ldo work mode
PMU_LDO_NORMAL	LDO normal work when pmu enter deepsleep mode 1
PMU_LDO_LOWPOW ER	LDO work at low power mode when pmu enter deepsleep mode 1
Input parameter{in}	
lowdrive	low-driver mode
PMU_LOWDRIVER_E NABLE	low-driver mode enable in deep-sleep 1 mode
PMU_LOWDRIVER_D ISABLE	low-driver mode disable in deep-sleep 1 mode
Input parameter{in}	
deepsleepmode1cmd	command to enter deepsleep mode 1
WFI_CMD	use WFI command
WFE_CMD	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PMU work in deepsleep mode 1 */

pmu_to_deepsleepmode_1(PMU_LDO_NORMAL,           PMU_LOWDRIVER_ENABLE,
WFI_CMD);
```

pmu_to_deepsleepmode_2

The description of pmu_to_deepsleepmode_2 is shown as below:

Table 3-699. Function pmu_to_deepsleepmode_2

Function name	pmu_to_deepsleepmode_2
Function prototype	void pmu_to_deepsleepmode_2(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmode2cmd);
Function descriptions	PMU work in deepsleep mode 2
Precondition	-
The called functions	-
Input parameter{in}	
ldo	ldo work mode

<i>PMU_LDO_NORMAL</i>	LDO normal work when pmu enter deepsleep mode 2
<i>PMU_LDO_LOWPOWER</i>	LDO work at low power mode when pmu enter deepsleep mode 2
Input parameter{in}	
lowdrive	low-driver mode
<i>PMU_LOWDRIVER_ENABLE</i>	low-driver mode enable in deep-sleep 2 mode
<i>PMU_LOWDRIVER_DISABLE</i>	low-driver mode disable in deep-sleep 2 mode
Input parameter{in}	
<i>deepsleepmode2cmd</i>	command to enter deepsleep mode 2
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PMU work in deepsleep mode 2 */

pmu_to_deepsleepmode_2(PMU_LDO_NORMAL,          PMU_LOWDRIVER_ENABLE,
WFI_CMD);
```

pmu_to_standbymode

The description of pmu_to_standbymode is shown as below:

Table 3-700. Function pmu_to_standbymode

Function name	pmu_to_standbymode
Function prototype	void pmu_to_standbymode(void);
Function descriptions	pmu work in standby mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PMU work in standby mode */

pmu_to_standby();
```

pmu_backup_write_enable

The description of pmu_backup_write_enable is shown as below:

Table 3-701. Function pmu_backup_write_enable

Function name	pmu_backup_write_enable
Function prototype	void pmu_backup_write_enable (void);
Function descriptions	enable backup domain write
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable backup domain write */
pmu_backup_write_enable();
```

pmu_backup_write_disable

The description of pmu_backup_write_disable is shown as below:

Table 3-702. Function pmu_backup_write_disable

Function name	pmu_backup_write_disable
Function prototype	void pmu_backup_write_disable (void);
Function descriptions	disable backup domain write
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable backup domain write */
pmu_backup_write_disable();
```

pmu_wakeup_pin_enable

The description of pmu_wakeup_pin_enable is shown as below:

Table 3-703. Function pmu_wakeup_pin_enable

Function name	pmu_wakeup_pin_enable
Function prototype	void pmu_wakeup_pin_enable(uint32_t wakeup_pin);
Function descriptions	enable wakeup pin
Precondition	-
The called functions	-
Input parameter{in}	
wakeup_pin	the pin to wakeup system
<i>PMU_WAKEUP_PIN0</i>	WKUP Pin 0 (PA0)
<i>PMU_WAKEUP_PIN1</i>	WKUP Pin 1 (PC13)
<i>PMU_WAKEUP_PIN2</i>	WKUP Pin 2 (PE6)
<i>PMU_WAKEUP_PIN3</i>	WKUP Pin 3 (PA2)
<i>PMU_WAKEUP_PIN4</i>	WKUP Pin 4 (PC5)
<i>PMU_WAKEUP_PIN5</i>	WKUP Pin 5 (PB5)
<i>PMU_WAKEUP_PIN6</i>	WKUP Pin 6 (PB15)
<i>PMU_WAKEUP_PIN7</i>	WKUP Pin 7 (PF8)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable wakeup pin 0 */
pmu_wakeup_pin_enable (PMU_WAKEUP_PIN0);
```

pmu_wakeup_pin_disable

The description of pmu_wakeup_pin_disable is shown as below:

Table 3-704. Function pmu_wakeup_pin_disable

Function name	pmu_wakeup_pin_disable
Function prototype	void pmu_wakeup_pin_disable (uint32_t wakeup_pin);
Function descriptions	disable wakeup pin
Precondition	-
The called functions	-
Input parameter{in}	
wakeup_pin	the pin to wakeup system
<i>PMU_WAKEUP_PIN0</i>	WKUP Pin 0 (PA0)
<i>PMU_WAKEUP_PIN1</i>	WKUP Pin 1 (PC13)
<i>PMU_WAKEUP_PIN2</i>	WKUP Pin 2 (PE6)

<i>PMU_WAKEUP_PIN3</i>	WKUP Pin 3 (PA2)
<i>PMU_WAKEUP_PIN4</i>	WKUP Pin 4 (PC5)
<i>PMU_WAKEUP_PIN5</i>	WKUP Pin 5 (PB5)
<i>PMU_WAKEUP_PIN6</i>	WKUP Pin 6 (PB15)
<i>PMU_WAKEUP_PIN7</i>	WKUP Pin 7 (PF8)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable wakeup pin 0 */
pmu_wakeup_pin_disable (PMU_WAKEUP_PIN0);
```

pmu_flag_clear

The description of pmu_flag_clear is shown as below:

Table 3-705. Function pmu_flag_clear

Function name	pmu_flag_clear
Function prototype	void pmu_flag_clear(uint32_t flag);
Function descriptions	clear flag bit
Precondition	-
The called functions	-
Input parameter{in}	
flag	flag
<i>PMU_FLAG_RESET_WAKEUP</i>	reset wakeup flag
<i>PMU_FLAG_RESET_STANDBY</i>	reset standby flag
<i>PMU_FLAG_RESET_DEEPSLEEP_1</i>	reset deep-sleep 1 mode status flag
<i>PMU_FLAG_RESET_DEEPSLEEP_2</i>	reset deep-sleep 2 mode status flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear flag bit */
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

pmu_flag_get

The description of pmu_flag_get is shown as below:

Table 3-706. Function pmu_flag_get

Function name	pmu_flag_get
Function prototype	FlagStatus pmu_flag_get(uint32_t flag);
Function descriptions	get flag state
Precondition	-
The called functions	-
Input parameter{in}	
flag	flag
<i>PMU_FLAG_WAKEUP</i>	wakeup flag
<i>PMU_FLAG_STANDBY</i>	standby flag
<i>PMU_FLAG_LVD</i>	low voltage detector status flag
<i>PMU_FLAG_HDRF</i>	high-driver ready flag
<i>PMU_FLAG_HDSRF</i>	high-driver switch ready flag
<i>PMU_FLAG_LDRF</i>	low-driver mode ready flag
<i>PMU_FLAG_DEEPSLE EP_1</i>	deep-sleep 1 mode status flag
<i>PMU_FLAG_DEEPSLE EP_2</i>	deep-sleep 2 mode status flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get flag state */

FlagStatus status;

status = pmu_flag_get (PMU_FLAG_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 3-707. 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_AHBEN	AHB 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_AHBRST	AHB reset register(only for CL、EPRT、E508 series)
RCU_CFG1	clock configuration register 1
RCU_DSV	deep-sleep mode voltage register
RCU_ADDCTL	additional clock control register
RCU_ADDCFG	additional clock configuration register(only for CL、E508 series)
RCU_ADDINT	additional clock interrupt register
RCU_PLLSSCTL	PLL clock spread spectrum control register(only for CL、EPRT、E508 series)
RCU_CFG2	clock configuration register 2
RCU_ADDAPB1RST	APB1 additional reset register
RCU_ADDAPB1EN	APB1 additional enable register

3.21.2. Descriptions of Peripheral functions

RCU firmware function are listed in the table shown as below:

Table 3-708. 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
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

Function name	Function description
rcu_apb2_clock_config	configure the APB2 clock prescaler selection
rcu_ckout0_config	configure the CK_OUT0 clock source
rcu_pll_config	configure the main PLL clock
rcu_pllpresel_config	configure the PLL clock source preselection
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_pllusbpresel_config	configure the PLLUSB clock source preselection(only for CL、E508 series)
rcu_pllusbpredv_config	configure the PLLUSBPREDV division factor and clock source(only for CL、E508 series)
rcu_pllusb_config	configure the PLLUSB clock(only for CL、E508 series)
rcu_adc_clock_config	configure the ADC prescaler factor
rcu_usb_clock_config	configure the USB prescaler factor
rcu_rtc_clock_config	configure the RTC clock source selection
rcu_shrtimer_clock_config	configure the SHRTIMER clock source selection(except for the EPRT series)
rcu_usart5_clock_config	configure the usart5 clock source selection
rcu_i2c2_clock_config	configure the I2C2 clock source selection
rcu_i2s1_clock_config	configure the I2S1 clock source selection(only for CL、EPRT、E508 series)
rcu_i2s2_clock_config	configure the I2S2 clock source selection(only for CL、EPRT、E508 series)
rcu_ck48m_clock_config	configure the CK48M clock selection
rcu_usbhssel_config	configure the USBHSEL source clock selection(only for CL、E508 series)
rcu_usbdv_config	configure the USBHSDV division factor(only for CL、E508 series)
rcu_lxtal_drive_capability_config	configure the LXTAL drive capability
rcu_osc_stab_wait	wait for oscillator stabilization flags is SET or oscillator startup is timeout
rcu_osc_on	turn on the oscillator
rcu_osc_off	turn off the oscillator
rcu_osc_bypass_mode_enable	enable the oscillator bypass mode
rcu_osc_bypass_mode_disable	disable the oscillator bypass mode
rcu_irc8m_adjust_value_set	set the IRC8M adjust value
rcu_hxtal_clock_monitor_enable	enable the HXTAL clock monitor
rcu_hxtal_clock_monitor_disable	disable the HXTAL clock monitor
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

Function name	Function description
rcu_all_reset_flag_clear	clear all the reset flag
rcu_interrupt_flag_get	get the clock stabilization interrupt and ckm flags
rcu_interrupt_flag_clear	clear the interrupt flags
rcu_interrupt_enable	enable the stabilization interrupt
rcu_interrupt_disable	disable the stabilization interrupt

Enum rcu_periph_enum

Table 3-709. Enum rcu_periph_enum

enum name	Function description
RCU_DMA0	DMA0 clock
RCU_DMA1	DMA1 clock
RCU_CRC	CRC clock
RCU_EXMC	EXMC clock
RCU_SDIO	SDIO clock(only for HD series)
RCU_USBHS	USBHS clock(only for CL、E508 series)
RCU_ULPI	ULPI clock(only for CL、E508 series)
RCU_ENET	ENET clock(only for CL、EPRT、E508 series)
RCU_ENETTX	ENETTX clock(only for CL、EPRT、E508 series)
RCU_ENETRX	ENETRX clock(only for CL、EPRT、E508 series)
RCU_TMU	TMU clock
RCU_SQPI	SQPI clock
RCU_TIMER1	TIMER1 clock
RCU_TIMER2	TIMER2 clock
RCU_TIMER3	TIMER3 clock
RCU_TIMER4	TIMER4 clock
RCU_TIMER5	TIMER5 clock
RCU_TIMER6	TIMER6 clock
RCU_TIMER11	TIMER11 clock(except for EPRT series)
RCU_TIMER12	TIMER12 clock(except for EPRT series)
RCU_TIMER13	TIMER13 clock(except for EPRT series)
RCU_WWDGT	WWDGT clock
RCU_SPI1	SPI1 clock
RCU_SPI2	SPI2 clock
RCU_USART1	USART1 clock
RCU_USART2	USART2 clock
RCU_UART3	UART3 clock
RCU_UART4	UART4 clock
RCU_I2C0	I2C0 clock
RCU_I2C1	I2C1 clock
RCU_USBD	USBD clock(only for HD、EPRT series)
RCU_I2C2	I2C2 clock

enum name	Function description
RCU_CAN0	CAN0 clock(except for EPRT series)
RCU_CAN1	CAN1 clock(except for EPRT series)
RCU_BKPI	BKPI clock
RCU_PMU	PMU clock
RCU_DAC	DAC clock
RCU_RTC	RTC clock
RCU_CTC	CTC clock
RCU_CAN2	CAN2 clock(only for CL、E508 series)
RCU_AF	alternate function clock
RCU_GPIOA	GPIOA clock
RCU_GPIOB	GPIOB clock
RCU_GPIOC	GPIOC clock
RCU_GPIOD	GPIOD clock
RCU_GPIOE	GPIOE clock
RCU_GPIOF	GPIOF clock
RCU_GPIOG	GPIOG clock
RCU_ADC0	ADC0 clock
RCU_ADC1	ADC1 clock
RCU_TIMER0	TIMER0 clock
RCU_SPI0	SPI0 clock
RCU_TIMER7	TIMER7 clock
RCU_USART0	USART0 clock
RCU_ADC2	ADC2 clock(except for CL series)
RCU_TIMER8	TIMER8 clock(except for EPRT series)
RCU_TIMER9	TIMER9 clock(except for EPRT series)
RCU_TIMER10	TIMER10 clock(except for EPRT series)
RCU_SHRTIMER	SHRTIMER clock(except for EPRT series)
RCU_USART5	USART5 clock
RCU_CMP	CMP clock(only for CL、E508 series)

Enum rcu_periph_sleep_enum

Table 3-710. Enum rcu_periph_sleep_enum

enum name	Function description
RCU_SRAM_SLP	SRAM clock when sleep mode
RCU_FMC_SLP	FMC clock when sleep mode

Enum rcu_periph_reset_enum

Table 3-711. Enum rcu_periph_reset_enum

enum name	Function description
RCU_USBHSRST	USBHS clock reset(only for CL、E508 series)

enum name	Function description
RCU_ENETRST	ENET clock reset(only for CL、EPRT、E508 series)
RCU_TMRURST	TMU clock reset
RCU_SQPIRST	SQPI clock reset
RCU_TIMER1RST	TIMER1 clock reset
RCU_TIMER2RST	TIMER2 clock reset
RCU_TIMER3RST	TIMER3 clock reset
RCU_TIMER4RST	TIMER4 clock reset
RCU_TIMER5RST	TIMER5 clock reset
RCU_TIMER6RST	TIMER6 clock reset
RCU_TIMER11RST	TIMER11 clock reset(except for EPRT series)
RCU_TIMER12RST	TIMER12 clock reset(except for EPRT series)
RCU_TIMER13RST	TIMER13 clock reset(except for EPRT series)
RCU_WWDGTRST	WWDGT clock reset
RCU_SPI1RST	SPI1 clock reset
RCU_SPI2RST	SPI2 clock reset
RCU_USART1RST	USART1 clock reset
RCU_USART2RST	USART2 clock reset
RCU_UART3RST	UART3 clock reset
RCU_USART4RST	UART4 clock reset
RCU_I2C0RST	I2C0 clock reset
RCU_I2C1RST	I2C1 clock reset
RCU_USBDRST	USBD clock reset(only for HD、EPRT series)
RCU_I2C2RST	I2C2 clock reset
RCU_CAN0RST	CAN0 clock reset(except for EPRT series)
RCU_CAN1RST	CAN1 clock reset(except for EPRT series)
RCU_BKPIRST	BKPI clock reset
RCU_PMURST	PMU clock reset
RCU_DACRST	DAC clock reset
RCU_CTCRST	CTC clock reset
RCU_CAN2RST	CAN2 clock reset(only for CL、E508 series)
RCU_AFRST	alternate function clock reset
RCU_GPIOARST	GPIOA clock reset
RCU_GPIOBRST	GPIOB clock reset
RCU_GPIOCRST	GPIOC clock reset
RCU_GPIODRST	GPIOD clock reset
RCU_GPIOERST	GPIOE clock reset
RCU_GPIOFRST	GPIOF clock reset
RCU_GPIOGRST	GPIOG clock reset
RCU_ADC0RST	ADC0 clock reset
RCU_ADC1RST	ADC1 clock reset
RCU_TIMER0RST	TIMER0 clock reset

enum name	Function description
RCU_SPI0RST	SPI0 clock reset
RCU_TIMER7RST	TIMER7 clock reset
RCU_USART0RST	USART0 clock reset
RCU_ADC2RST	ADC2 clock reset(except for CL series)
RCU_TIMER8RST	TIMER8 clock reset(except for EPRT series)
RCU_TIMER9RST	TIMER9 clock reset(except for EPRT series)
RCU_TIMER10RST	TIMER10 clock reset(except for EPRT series)
RCU_USART5RST	USART5 clock reset
RCU_SHRTIMERRST	HPTIEMR clock reset(except for EPRT series)
RCU_CMPRST	CMP clock reset(only for CL、E508 series)

Enum rcu_flag_enum

Table 3-712. Enum rcu_flag_enum

enum name	Function description
RCU_FLAG_IRC8MSTB	IRC8M stabilization flags
RCU_FLAG_HXTALSTB	HXTAL stabilization flags
RCU_FLAG_PLLSTB	PLL stabilization flags
RCU_FLAG_PLL1STB	PLL1 stabilization flags(only for CL、EPRT、E508 series)
RCU_FLAG_PLL2STB	PLL2 stabilization flags(only for CL、EPRT、E508 series)
RCU_FLAG_PLLUSBSTB	PLLUSB stabilization flags(only for CL、E508 series)
RCU_FLAG_LXTALSTB	LXTAL stabilization flags
RCU_FLAG_IRC40KSTB	IRC40K stabilization flags
RCU_FLAG_IRC48MSTB	IRC48M stabilization flags
RCU_FLAG_BORRST	BOR reset flag
RCU_FLAG_EPRST	external PIN reset flags
RCU_FLAG_PORRST	power reset flags
RCU_FLAG_SWRST	software reset flags
RCU_FLAG_FWDGTRST	FWDGT reset flags
RCU_FLAG_WWDGTRST	WWDGT reset flags
RCU_FLAG_LPRST	low-power reset flags

Enum rcu_int_flag_enum
Table 3-713. Enum rcu_int_flag_enum

enum name	Function description
RCU_INT_FLAG_IRC4_0KSTB	IRC40K stabilization interrupt flag
RCU_INT_FLAG_LXT_ALSTB	LXTAL stabilization interrupt flag
RCU_INT_FLAG_IRC8_MSTB	IRC8M stabilization interrupt flag
RCU_INT_FLAG_HXT_ALSTB	HXTAL stabilization interrupt flag
RCU_INT_FLAG_PLL_STB	PLL stabilization interrupt flag
RCU_INT_FLAG_PLL_1STB	PLL1 stabilization interrupt flag(only for CL、EPRT、E508 series)
RCU_INT_FLAG_PLL_2STB	PLL2 stabilization interrupt flag(only for CL、EPRT、E508 series)
RCU_INT_FLAG_PLL_USBSTB	PLLUSB stabilization interrupt flag(only for CL、E508 series)
RCU_INT_FLAG_CKM	HXTAL clock stuck interrupt flag
RCU_INT_FLAG_IRC4_8MSTB	IRC48M stabilization interrupt flag

Enum rcu_int_flag_clear_enum
Table 3-714. Enum rcu_int_flag_clear_enum

enum name	Function description
RCU_INT_FLAG_IRC4_0KSTB_CLR	IRC40K stabilization interrupt flags clear
RCU_INT_FLAG_LXT_ALSTB_CLR	LXTAL stabilization interrupt flags clear
RCU_INT_FLAG_IRC8_MSTB_CLR	IRC8M stabilization interrupt flags clear
RCU_INT_FLAG_HXT_ALSTB_CLR	HXTAL stabilization interrupt flags clear
RCU_INT_FLAG_PLL_STB_CLR	PLL stabilization interrupt flags clear
RCU_INT_FLAG_PLL_1STB_CLR	PLL1 stabilization interrupt flags clear(only for CL、EPRT、E508 series)
RCU_INT_FLAG_PLL_2STB_CLR	PLL2 stabilization interrupt flags clear(only for CL、EPRT、E508 series)
RCU_INT_FLAG_PLL_USBSTB_CLR	PLLUSB stabilization interrupt flags clear(only for CL、E508 series)

enum name	Function description
RCU_INT_FLAG_CKM_CLR	CKM interrupt flags clear
RCU_INT_FLAG_IRC48MSTB_CLR	internal 48 MHz RC oscillator stabilization interrupt clear

Enum rcu_int_enum

Table 3-715. Enum rcu_int_enum

enum name	Function description
RCU_INT_IRC40KSTB	IRC40K stabilization interrupt
RCU_INT_LXTALSTB	LXTAL stabilization interrupt
RCU_INT_IRC8MSTB	IRC8M stabilization interrupt
RCU_INT_HXTALSTB	HXTAL stabilization interrupt
RCU_INT_PLLSTB	PLL stabilization interrupt
RCU_INT_PLL1STB	PLL1 stabilization interrupt(only for CL、EPRT、E508 series)
RCU_INT_PLL2STB	PLL2 stabilization interrupt(only for CL、EPRT、E508 series)
RCU_INT_PLLUSBSTB	PLLUSB stabilization interrupt(only for CL、E508 series)
RCU_INT_IRC48MSTB	internal 48 MHz RC oscillator stabilization interrupt

Enum rcu_osc_type_enum

Table 3-716. Enum rcu_osc_type_enum

enum name	Function description
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL
RCU_IRC8M	IRC8M
RCU_IRC48M	IRC48M
RCU_IRC40K	IRC40K
RCU_PLL_CK	PLL
RCU_PLL1_CK	PLL1(only for CL、EPRT、E508 series)
RCU_PLL2_CK	PLL2(only for CL、EPRT、E508 series)
RCU_PLLUSB_CK	PLLUSB(only for CL、E508 series)

Enum rcu_clock_freq_enum

Table 3-717. Enum rcu_clock_freq_enum

enum name	Function description
CK_SYS	system clock
CK_AHB	AHB clock
CK_APB1	APB1 clock
CK_APB2	APB2 clock

enum name	Function description
CK_USART	USART5 clock

rcu_deinit

The description of rcu_deinit is shown as below:

Table 3-718. Function rcu_deinit

Function name	rcu_deinit
Function prototype	void rcu_deinit(void);
Function descriptions	deinitialize the RCU
Precondition	-
The called functions	-
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 3-719. 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 3-709. Enum rcu_periph_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the USART0 clock */
rcu_periph_clock_enable(RCU_USART0);
```

rcu_periph_clock_disable

The description of `rcu_periph_clock_disable` is shown as below:

Table 3-720. Function `rcu_periph_clock_disable`

Function name	rcu_periph_clock_disable
Function prototype	void <code>rcu_periph_clock_disable(rcu_periph_enum periph);</code>
Function descriptions	disable the peripherals clock
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to Table 3-709. Enum <code>rcu_periph_enum</code>
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 3-721. Function `rcu_periph_clock_sleep_enable`

Function name	rcu_periph_clock_sleep_enable
Function prototype	void <code>rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);</code>
Function descriptions	enable the peripherals clock when in sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to Table 3-710. Enum <code>rcu_periph_sleep_enum</code>
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 3-722. Function `rcu_periph_clock_sleep_disable`

Function name	rcu_periph_clock_sleep_disable
Function prototype	void <code>rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);</code>
Function descriptions	disable the peripherals clock when in sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to Table 3-710. Enum <code>rcu_periph_sleep_enum</code>
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 3-723. Function `rcu_periph_reset_enable`

Function name	rcu_periph_reset_enable
Function prototype	void <code>rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);</code>
Function descriptions	enable the peripherals reset
Precondition	-
The called functions	-
Input parameter{in}	
periph_reset	RCU peripherals reset, refer to Table 3-711. Enum <code>rcu_periph_reset_enum</code>
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 3-724. Function `rcu_periph_reset_disable`

Function name	rcu_periph_reset_disable
Function prototype	void <code>rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);</code>
Function descriptions	disable the peripheral reset
Precondition	-
The called functions	-
Input parameter{in}	
<code>periph_reset</code>	RCU peripherals reset, refer to Table 3-711. Enum <code>rcu_periph_reset_enum</code>
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 3-725. Function `rcu_bkp_reset_enable`

Function name	rcu_bkp_reset_enable
Function prototype	void <code>rcu_bkp_reset_enable(void);</code>
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 3-726. 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 3-727. Function rcu_system_clock_source_config

Function name	rcu_system_clock_source_config
Function prototype	void rcu_system_clock_source_config(uint32_t ck_sys);
Function descriptions	configure the system clock source
Precondition	-
The called functions	-
Input parameter{in}	
ck_sys	system clock source select
<i>RCU_CKSYSRC_IRC_8M</i>	select CK_IRC8M as the CK_SYS source
<i>RCU_CKSYSRC_HXTAL</i>	select CK_HXTAL as the CK_SYS source
<i>RCU_CKSYSRC_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_CKSYSRC_HXTAL);

```

rcu_system_clock_source_get

The description of `rcu_system_clock_source_get` is shown as below:

Table 3-728. 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	RCU_SCSS_IRC8M / RCU_SCSS_HXTAL / RCU_SCSS_PLL

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 3-729. 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
RCU_AHB_CKSYS_DI Vx	select CK_SYS / x as CK_AHB, (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 3-730. 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_DIV1</i>	select CK_AHB as CK_APB1
<i>RCU_APB1_CKAHB_DIV2</i>	select CK_AHB / 2 as CK_APB1
<i>RCU_APB1_CKAHB_DIV4</i>	select CK_AHB / 4 as CK_APB1
<i>RCU_APB1_CKAHB_DIV8</i>	select CK_AHB / 8 as CK_APB1
<i>RCU_APB1_CKAHB_DIV16</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 3-731. Function rcu_apb2_clock_config

Function name	rcu_apb2_clock_config
Function prototype	void rcu_apb2_clock_config(uint32_t ck_apb2);
Function descriptions	configure the APB2 clock prescaler selection

Precondition	-
The called functions	-
Input parameter{in}	
ck_apb2	APB2 clock prescaler selection
<i>RCU_APB2_CKAHB_DIV1</i>	select CK_AHB as CK_APB2
<i>RCU_APB2_CKAHB_DIV2</i>	select CK_AHB / 2 as CK_APB2
<i>RCU_APB2_CKAHB_DIV4</i>	select CK_AHB / 4 as CK_APB2
<i>RCU_APB2_CKAHB_DIV8</i>	select CK_AHB / 8 as CK_APB2
<i>RCU_APB2_CKAHB_DIV16</i>	select CK_AHB / 16 as CK_APB2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CK_AHB / 8 as CK_APB2 */
rcu_apb2_clock_config(RCU_APB2_CKAHB_DIV8);
```

rcu_ckout0_config

The description of `rcu_ckout0_config` is shown as below:

Table 3-732. Function `rcu_ckout0_config`

Function name	<code>rcu_ckout0_config</code>
Function prototype	<code>void rcu_ckout0_config(uint32_t ckout0_src);</code>
Function descriptions	configure the CK_OUT0 clock source
Precondition	-
The called functions	-
Input parameter{in}	
ckout0_src	CK_OUT0 clock source selection
<i>RCU_CKOUT0SRC_NONE</i>	no clock selected
<i>RCU_CKOUT0SRC_CKSYS</i>	system clock selected
<i>RCU_CKOUT0SRC_IRC8M</i>	high speed 8M internal oscillator clock selected
<i>RCU_CKOUT0SRC_HXTAL</i>	HXTAL selected

<i>RCU_CKOUT0SRC_C_KPLL_DIV2</i>	CK_PLL / 2 selected
<i>RCU_CKOUT0SRC_C_KPLL1</i>	CK_PLL1 selected (only available for CL and EPRT series)
<i>RCU_CKOUT0SRC_C_KPLL2_DIV2</i>	CK_PLL2 / 2 selected (only available for CL and EPRT series)
<i>RCU_CKOUT0SRC_C_KPLL2</i>	EXT1 selected (only available for CL and EPRT series)
<i>RCU_CKOUT0SRC_E_XT1</i>	CK_PLL2 clock selected (only available for CL and EPRT series)
<i>RCU_CKOUT0SRC_C_KIRC48M</i>	CK_IRC48M clock selected (only available for CL and EPRT series)
<i>RCU_CKOUT0SRC_C_KIRC48M_DIV8</i>	CK_IRC48M / 8 clock selected (only available for CL and EPRT series)
<i>RCU_CKOUT0SRC_C_KPLLUSB_DIV32</i>	CK_PLLUSB / 32 clock selected (only available for CL series)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the HXTAL as CK_OUT0 clock source */
rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL);
```

rcu_pll_config

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

Table 3-733. 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_IRC48M</i>	HXTAL or IRC48M is selected as source clock of PLL
Input parameter{in}	
pll_mul	PLL clock multiplication factor

<i>RCU_PLL_MULx</i>	PLL clock * x (HD series x = 2..63, CL series x = 2..14, 16..64, 6.5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLL */
rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PLL_MUL10);
```

rcu_pllpresel_config

The description of `rcu_pllpresel_config` is shown as below:

Table 3-734. Function `rcu_pllpresel_config`

Function name	<code>rcu_pllpresel_config</code>
Function prototype	<code>void rcu_pllpresel_config(uint32_t pll_presel);</code>
Function descriptions	configure the PLL clock source preselection
Precondition	-
The called functions	-
Input parameter{in}	
<code>pll_presel</code>	PLL clock source preselection
<code>RCU_PLLPRESRC_HXTAL</code>	HXTAL selected as PREDV0 input source clock
<code>RCU_PLLPRESRC_IRC48M</code>	CK_PLL selected as PREDV0 input source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLL clock source preselection */
rcu_pllpresel_config (RCU_PLLPRESRC_HXTAL);
```

rcu_predv0_config (HD series)

The description of `rcu_predv0_config` is shown as below:

Table 3-735. Function `rcu_predv0_config`

Function name	<code>rcu_predv0_config</code>
Function prototype	<code>void rcu_predv0_config(uint32_t predv0_div);</code>
Function descriptions	configure the PREDV0 division factor
Precondition	-

The called functions		-
Input parameter{in}		
predv0_div		PREDV0 division factor
<i>RCU_PREDV0_DIVx</i>		PREDV0 input source clock is divided x (x = 1, 2)
Output parameter{out}		
-		-
Return value		
-		-

Example:

```
/* configure the PREDV0 division factor */

rcu_predv0_config(RCU_PREDV0_DIV1);
```

rcu_predv0_config (CL、EPRT、E508 series)

The description of `rcu_predv0_config` is shown as below:

Table 3-736. Function `rcu_predv0_config`

Function name	rcu_predv0_config
Function prototype	void <code>rcu_predv0_config(uint32_t predv0_source, uint32_t predv0_div);</code>
Function descriptions	configure the PREDV0 division factor
Precondition	-
The called functions	-
Input parameter{in}	
predv0_source	PREDV0 input clock source selection
<i>RCU_PREDV0SRC_H_XTAL_IRC48M</i>	HXTAL or IRC48M selected as PREDV0 input source clock
<i>RCU_PREDV0SRC_C_KPLL1</i>	CK_PLL1 selected 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_IRC48M, RCU_PREDV0_DIV);
```

rcu_predv1_config

The description of rcu_predv1_config is shown as below:

Table 3-737. 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 3-738. 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
<i>RCU_PLL1_MULx</i>	PLL1 clock * x, (x = 8..14, 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 3-739. 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
<i>RCU_PLL2_MULx</i>	PLL2 clock * x, (x = 8..14, 16, 20, 18..32, 40, 34..64, 80)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLL2 clock */
rcu_pll2_config(RCU_PLL2_MUL8);
```

rcu_pllusbpresel_config

The description of rcu_pllusbpresel_config is shown as below:

Table 3-740. Function rcu_pllusbpresel_config

Function name	rcu_pllusbpresel_config
Function prototype	void rcu_pllusbpresel_config(uint32_t pllusb_presel);
Function descriptions	configure the PLLUSB clock source preselection
Precondition	-
The called functions	-
Input parameter{in}	
pllusb_presel	PLLUSB clock source preselection
<i>RCU_PLLUSBPRESR_C_HXTAL</i>	HXTAL selected as PLLUSB source clock
<i>RCU_PLLUSBPRESR_C_IRC48M</i>	IRC48M clock selected as PLLUSB source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLLUSB clock source preselection */
```

```
rcu_pllusbpresel_config(RCU_PLLUSBPRESRC_HXTAL);
```

rcu_pllusbpredv_config

The description of `rcu_pllusbpredv_config` is shown as below:

Table 3-741. Function `rcu_pllusbpredv_config`

Function name	rcu_pllusbpredv_config
Function prototype	void rcu_pllusbpredv_config(uint32_t pllusbpredv_source, uint32_t pllusbpredv_div);
Function descriptions	configure the PLLUSBPREDV division factor and clock source
Precondition	-
The called functions	-
Input parameter{in}	
pllusbpredv_source	PLLUSBPREDV input clock source selection
<i>RCU_PLLUSBPREDVS</i> <i>RC_HXTAL_IRC48M</i>	HXTAL or IRC48M selected as PLLUSBPREDV input source clock
<i>RCU_PLLUSBPREDVS</i> <i>RC_CKPLL1</i>	CK_PLL1 selected as PLLUSBPREDV input source clock
Input parameter{in}	
pllusbpredv_div	PLLUSBPREDV division factor
<i>RCU_PLLUSBPREDV_</i> <i>DIVx</i>	PLLUSBPREDV input source clock divided by x, (x = 1..15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLLUSBPREDV division factor and clock source */
```

```
rcu_pllusbpredv_config(RCU_PLLUSBPREDVSRC_HXTAL_IRC48M, RCU_PLLUSBPREDV_DIV15);
```

rcu_pllusb_config

The description of `rcu_pllusb_config` is shown as below:

Table 3-742. Function `rcu_pllusb_config`

Function name	rcu_pllusb_config
Function prototype	void rcu_pllusb_config(uint32_t pllusb_mul);
Function descriptions	configure the PLLUSB clock
Precondition	-
The called functions	-
Input parameter{in}	

pllusb_mul	PLLUSB clock multiplication factor
<i>RCU_PLLUSB_MULx</i>	PLLUSB source clock multiply by x, (x = 16, 17..127)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLLUSB clock */
rcu_pllusb_config(RCU_PLLUSB_MUL16);
```

rcu_adc_clock_config

The description of `rcu_adc_clock_config` is shown as below:

Table 3-743. Function `rcu_adc_clock_config`

Function name	<code>rcu_adc_clock_config</code>
Function prototype	<code>void rcu_adc_clock_config(uint32_t adc_psc);</code>
Function descriptions	configure the ADC prescaler factor
Precondition	-
The called functions	-
Input parameter{in}	
adc_psc	ADC prescaler factor
<i>RCU_CKADC_CKAPB_2_DIV2</i>	ADC prescaler select CK_APB2 / 2
<i>RCU_CKADC_CKAPB_2_DIV4</i>	ADC prescaler select CK_APB2 / 4
<i>RCU_CKADC_CKAPB_2_DIV6</i>	ADC prescaler select CK_APB2 / 6
<i>RCU_CKADC_CKAPB_2_DIV8</i>	ADC prescaler select CK_APB2 / 8
<i>RCU_CKADC_CKAPB_2_DIV12</i>	ADC prescaler select CK_APB2 / 12
<i>RCU_CKADC_CKAPB_2_DIV16</i>	ADC prescaler select CK_APB2 / 16
<i>RCU_CKADC_CKAHB_DIV5</i>	ADC prescaler select CK_AHB / 5
<i>RCU_CKADC_CKAHB_DIV6</i>	ADC prescaler select CK_AHB / 6
<i>RCU_CKADC_CKAHB_DIV10</i>	ADC prescaler select CK_AHB / 10
<i>RCU_CKADC_CKAHB_DIV20</i>	ADC prescaler select CK_AHB / 20

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the ADC prescaler factor */

rcu_adc_clock_config(RCU_CKADC_CKAPB2_DIV8);
```

rcu_usb_clock_config

The description of `rcu_usb_clock_config` is shown as below:

Table 3-744. Function `rcu_usb_clock_config`

Function name	rcu_usb_clock_config
Function prototype	void rcu_usb_clock_config(uint32_t usb_psc);
Function descriptions	configure the USB prescaler factor
Precondition	-
The called functions	-
Input parameter{in}	
usb_psc	USB prescaler factor
<i>RCU_CKUSB_CKPLL_DIV1_5</i>	USBD / USBHS prescaler select CK_PLL / 1.5
<i>RCU_CKUSB_CKPLL_DIV1</i>	USBD / USBHS prescaler select CK_PLL / 1
<i>RCU_CKUSB_CKPLL_DIV2_5</i>	USBD / USBHS prescaler select CK_PLL / 2.5
<i>RCU_CKUSB_CKPLL_DIV2</i>	USBD / USBHS prescaler select CK_PLL / 2
<i>RCU_CKUSB_CKPLL_DIV3</i>	USBD / USBHS prescaler select CK_PLL / 3
<i>RCU_CKUSB_CKPLL_DIV3_5</i>	USBD / USBHS prescaler select CK_PLL / 3.5
<i>RCU_CKUSB_CKPLL_DIV4</i>	USBD / USBHS prescaler select CK_PLL / 4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the USB prescaler factor */

rcu_usb_clock_config(RCU_CKUSB_CKPLL_DIV2_5);
```

rcu_rtc_clock_config

The description of rcu_rtc_clock_config is shown as below:

Table 3-745. 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_shrtimer_clock_config

The description of rcu_shrtimer_clock_config is shown as below:

Table 3-746. Function rcu_shrtimer_clock_config

Function name	rcu_shrtimer_clock_config
Function prototype	void rcu_shrtimer_clock_config (uint32_t shrtimer_clock_source);
Function descriptions	configure the SHRTIMER clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
shrtimer_clock_source	SHRTIMER clock source selection
<i>RCU_SHRTIMERSRC_CKAPB2</i>	APB2 clock selected as SHRTIMER source clock
<i>RCU_SHRTIMERSRC_CKSYS</i>	system clock selected as SHRTIMER source clock

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the SHRTIMER clock source selection */

rcu_shrtimer_clock_config (RCU_SHRTIMERSRC_CKAPB2);
```

rcu_usart5_clock_config

The description of `rcu_usart5_clock_config` is shown as below:

Table 3-747. Function `rcu_usart5_clock_config`

Function name	rcu_usart5_clock_config
Function prototype	void rcu_usart5_clock_config(uint32_t usart5_clock_source);
Function descriptions	configure the USART5 clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart5_clock_source</code>	USART5 clock source selection
<code>RCU_USART5SRC_C_KAPB2</code>	APB2 clock selected as USART5 source clock
<code>RCU_USART5SRC_C_KSYS</code>	system clock selected as USART5 source clock
<code>RCU_USART5SRC_LX_TAL</code>	LXTAL clock selected as USART5 source clock
<code>RCU_USART5SRC_IR_C8M</code>	IRC8M clock selected as USART5 source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the USART5 clock source selection */

rcu_usart5_clock_config(RCU_USART5SRC_CKAPB2);
```

rcu_i2c2_clock_config

The description of `rcu_i2c2_clock_config` is shown as below:

Table 3-748. Function `rcu_i2c2_clock_config`

Function name	rcu_i2c2_clock_config
---------------	-----------------------

Function prototype	void rcu_i2c2_clock_config(uint32_t i2c2_clock_source);
Function descriptions	configure the I2C2 clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
i2c2_clock_source	I2C2 clock source selection
<i>RCU_I2C2SRC_CKAP_B1</i>	APB1 clock selected as I2C2 source clock
<i>RCU_I2C2SRC_CKSYS</i>	System clock selected as I2C2 source clock
<i>RCU_I2C2SRC_SRC_C_KIRC8M</i>	CK_IRC8M clock selected as I2C2 source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the I2C2 clock source selection */
rcu_i2c2_clock_config(RCU_I2C2SRC_CKAPB1);
```

rcu_ck48m_clock_config

The description of **rcu_ck48m_clock_config** is shown as below:

Table 3-749. Function **rcu_ck48m_clock_config**

Function name	rcu_ck48m_clock_config
Function prototype	void rcu_ck48m_clock_config(uint32_t ck48m_clock_source);
Function descriptions	configure the CK48M clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
ck48m_clock_source	CK48M clock source selection
<i>RCU_CK48MSRC_CK_PLL</i>	CK_PLL selected as CK48M source clock
<i>RCU_CK48MSRC_IRC48M</i>	CK_IRC48M selected as CK48M source clock
<i>RCU_CK48MSRC_CK_PLLUSB</i>	CKPLLUSB selected as CK48M source clock
<i>RCU_CK48MSRC_CK_PLL2</i>	CKPLL2 selected as CK48M source clock
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure the CK48M clock source selection */
rcu_ck48m_clock_config(RCU_CK48MSRC_IRC48M);
```

rcu_i2s1_clock_config

The description of `rcu_i2s1_clock_config` is shown as below:

Table 3-750. 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	I2S clock source selection
<i>RCU_I2S1SRC_CKSY</i> S	system clock selected as I2S1 source clock
<i>RCU_I2S1SRC_CKPLL</i> 2_MUL2	CK_PLL2 * 2 selected 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 3-751. 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	I2S clock source selection

<i>RCU_I2S2SRC_CKSYS</i>	system clock selected as I2S2 source clock
<i>RCU_I2S2SRC_CKPLL2_MUL2</i>	CK_PLL2 * 2 selected as I2S2 source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the I2S2 clock source selection */
rcu_i2s2_clock_config(RCU_I2S2SRC_CKPLL2_MUL2);
```

The description of `rcu_usbhssel_config` is shown as below:

Table 3-752. Function `rcu_usbhssel_config`

Function name	<code>rcu_usbhssel_config</code>
Function prototype	<code>void rcu_usbhssel_config(uint32_t usbhssel_clock_source);</code>
Function descriptions	configure the USBHSSEL source clock selection
Precondition	-
The called functions	-
Input parameter{in}	
usbhssel_clock_source	USBHSSEL clock source selection
<i>RCU_USBHSSRC_48M</i>	48M clock selected as USBHS source clock
<i>RCU_CK48MSRC_60M</i>	60M clock selected as USBHS source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the USBHSSEL source clock selection */
rcu_usbhssel_config(RCU_USBHSSRC_48M);
```

`rcu_usbdv_config`

The description of `rcu_usbdv_config` is shown as below:

Table 3-753. Function `rcu_usbdv_config`

Function name	<code>rcu_usbdv_config</code>
Function prototype	<code>void rcu_usbdv_config(uint32_t usbhs_dv);</code>
Function descriptions	configure the USBHSDV division factor

Precondition	-
The called functions	-
Input parameter{in}	
usbhs_dv	USBHSDV division factor
<i>RCU_USBHSDV_DIV2</i>	USBHSDV input source clock divided by 2
<i>RCU_USBHSDV_DIV4</i>	USBHSDV input source clock divided by 4
<i>RCU_USBHSDV_DIV6</i>	USBHSDV input source clock divided by 6
<i>RCU_USBHSDV_DIV8</i>	USBHSDV input source clock divided by 8
<i>RCU_USBHSDV_DIV10</i>	USBHSDV input source clock divided by 10
<i>RCU_USBHSDV_DIV12</i>	USBHSDV input source clock divided by 12
<i>RCU_USBHSDV_DIV14</i>	USBHSDV input source clock divided by 14
<i>RCU_USBHSDV_DIV16</i>	USBHSDV input source clock divided by 16
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the USBHSDV division factor */

rcu_usbdv_config(RCU_USBHSDV_DIV16);
```

rcu_lxtal_drive_capability_config

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

Table 3-754. Function `rcu_lxtal_drive_capability_config`

Function name	<code>rcu_lxtal_drive_capability_config</code>
Function prototype	<code>void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);</code>
Function descriptions	configure the LXTAL drive capability
Precondition	-
The called functions	-
Input parameter{in}	
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_GHDRI</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_oscி_stab_wait

The description of `rcu_oscி_stab_wait` is shown as below:

Table 3-755. Function `rcu_oscி_stab_wait`

Function name	rcu_oscி_stab_wait
Function prototype	ErrStatus rcu_oscி_stab_wait(rcu_oscி_type_enum oscி);
Function descriptions	wait for oscillator stabilization flags is SET or oscillator startup is timeout
Precondition	-
The called functions	-
Input parameter{in}	
oscி	oscillator types, refer to Table 3-716. Enum <code>rcu_oscி_type_enum</code>
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* wait for oscillator stabilization flag */
if(SUCCESS == rcu_oscி_stab_wait(RCU_HXTAL)){
}
```

rcu_oscி_on

The description of `rcu_oscி_on` is shown as below:

Table 3-756. Function `rcu_oscி_on`

Function name	rcu_oscி_on
Function prototype	void rcu_oscி_on(rcu_oscி_type_enum oscி);
Function descriptions	turn on the oscillator
Precondition	-
The called functions	-
Input parameter{in}	
oscி	oscillator types, refer to Table 3-716. Enum <code>rcu_oscி_type_enum</code>

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn on the high speed crystal oscillator */

rcu_oscı_on(RCU_HXTAL);
```

rcu_oscı_off

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

Table 3-757. Function **rcu_oscı_off**

Function name	rcu_oscı_off
Function prototype	void rcu_oscı_off(rcu_oscı_type_enum oscı);
Function descriptions	turn off the oscillator
Precondition	-
The called functions	-
Input parameter{in}	
oscı	oscillator types, refer to Table 3-716. Enum rcu_oscı_type_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn off the high speed crystal oscillator */

rcu_oscı_off(RCU_HXTAL);
```

rcu_oscı_bypass_mode_enable

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

Table 3-758. Function **rcu_oscı_bypass_mode_enable**

Function name	rcu_oscı_bypass_mode_enable
Function prototype	void rcu_oscı_bypass_mode_enable(rcu_oscı_type_enum oscı);
Function descriptions	enable the oscillator bypass mode
Precondition	HXTALEN or LXTALEN must be reset before it
The called functions	-
Input parameter{in}	
oscı	oscillator types, refer to Table 3-716. Enum rcu_oscı_type_enum
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the high speed crystal oscillator bypass mode */
rcu_osc_bypass_mode_enable(RCU_HXTAL);
```

rcu_osc_bypass_mode_disable

The description of `rcu_osc_bypass_mode_disable` is shown as below:

Table 3-759. Function `rcu_osc_bypass_mode_disable`

Function name	rcu_osc_bypass_mode_disable
Function prototype	void rcu_osc_bypass_mode_disable(rcu_osc_type_enum osci);
Function descriptions	disable the oscillator bypass mode
Precondition	HXTALEN or LXTALEN must be reset before it
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to Table 3-716. Enum <code>rcu_osc_type_enum</code>
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the high speed crystal oscillator bypass mode */
rcu_osc_bypass_mode_disable(RCU_HXTAL);
```

rcu_irc8m_adjust_value_set

The description of `rcu_irc8m_adjust_value_set` is shown as below:

Table 3-760. Function `rcu_irc8m_adjust_value_set`

Function name	rcu_irc8m_adjust_value_set
Function prototype	void rcu_irc8m_adjust_value_set(uint32_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_hxtal_clock_monitor_enable

The description of `rcu_hxtal_clock_monitor_enable` is shown as below:

Table 3-761. 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 3-762. 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_deepsleep_voltage_set

The description of `rcu_deepsleep_voltage_set` is shown as below:

Table 3-763. 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
<i>RCU_DEEPSLEEP_V_1_0</i>	the core voltage is 1.0V in deep-sleep mode
<i>RCU_DEEPSLEEP_V_0_9</i>	the core voltage is 0.9V in deep-sleep mode
<i>RCU_DEEPSLEEP_V_0_8</i>	the core voltage is 0.8V in deep-sleep mode
<i>RCU_DEEPSLEEP_V_0_7</i>	the core voltage is 0.7V in deep-sleep mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the deep-sleep mode voltage */
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1_0);
```

rcu_clock_freq_get

The description of `rcu_clock_freq_get` is shown as below:

Table 3-764. 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	Clock frequency, refers to Table 3-717. 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
<i>CK_USART</i>	USART5 clock frequency
Output parameter{out}	
-	-
Return value	
ck_freq	clock frequency of system, AHB, APB1, APB2, USART5

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 3-765. 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 3-712. Enum rcu_flag_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the clock stabilization flag */

if(RESET != rcu_flag_get(RCU_FLAG_LXTALSTB)){

}
```

rcu_all_reset_flag_clear

The description of rcu_all_reset_flag_clear is shown as below:

Table 3-766. Function rcu_all_reset_flag_clear

Function name	rcu_all_reset_flag_clear
Function prototype	void rcu_all_reset_flag_clear(void);
Function descriptions	clear all the reset flag
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear all the reset flag */
rcu_all_reset_flag_clear();
```

rcu_interrupt_flag_get

The description of rcu_interrupt_flag_get is shown as below:

Table 3-767. Function rcu_interrupt_flag_get

Function name	rcu_interrupt_flag_get
Function prototype	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);
Function descriptions	get the clock stabilization interrupt and ckm flags
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	interrupt and ckm flags, refer to Table 3-713. Enum rcu_int_flag_enum
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 3-768. 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);
Function descriptions	clear the interrupt flags
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	clock stabilization and stuck interrupt flags clear, refer to Table 3-714. Enum rcu_int_flag_clear_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the interrupt HXTAL stabilization interrupt */
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

rcu_interrupt_enable

The description of rcu_interrupt_enable is shown as below:

Table 3-769. Function rcu_interrupt_enable

Function name	rcu_interrupt_enable
Function prototype	void rcu_interrupt_enable(rcu_int_enum interrupt);
Function descriptions	enable the stabilization interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	clock stabilization interrupt, refer to Table 3-715. Enum rcu_int_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the HXTAL stabilization interrupt */
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

rcu_interrupt_disable

The description of rcu_interrupt_disable is shown as below:

Table 3-770. Function rcu_interrupt_disable

Function name	rcu_interrupt_disable
Function prototype	void rcu_interrupt_disable(rcu_int_enum interrupt);
Function descriptions	disable the stabilization interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	clock stabilization interrupt, refer to Table 3-715. Enum rcu_int_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

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 3-771. 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

Registers	Descriptions
RTC_ALRML	Alarm low register

3.22.2. Descriptions of Peripheral functions

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

Table 3-772. 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
rtc_prescaler_set	set RTC prescaler value
rtc_alarm_config	set RTC alarm value
rtc_divider_get	get RTC divider value
rtc_interrupt_enable	enable RTC interrupt
rtc_interrupt_disable	disable RTC interrupt
rtc_flag_get	get RTC flag status
rtc_flag_clear	clear RTC flag status

rtc_configuration_mode_enter

The description of rtc_configuration_mode_enter is shown as below:

Table 3-773. 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 3-774. 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 3-775. 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( );

/* enable the RTC second interrupt */

rtc_interrupt_enable(RTC_INT_SECOND);
```

rtc_register_sync_wait

The description of rtc_register_sync_wait is shown as below:

Table 3-776. 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 3-777. 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 3-778. 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	-
The called functions	-
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 3-779. Function rtc_prescaler_set

Function name	rtc_interrupt_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 0xFFFF */

  rtc_prescaler_set (0xFFFF);

```

rtc_alarm_config

The description of `rtc_alarm_config` is shown as below:

Table 3-780. 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 <code>rtc_lwoff_wait()</code> function (wait until LWOFF flag is set). -
The called functions	<code>rtc_configuration_mode_enter</code> / <code>rtc_configuration_mode_exit</code>
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 3-781. 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_interrupt_enable

The description of `rtc_interrupt_enable` is shown as below:

Table 3-782. 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 <code>rtc_lwoff_wait ()</code> 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 3-783. 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);
```

rtc_flag_get

The description of `rtc_flag_get` is shown as below:

Table 3-784. Function `rtc_flag_get`

Function name		<code>rtc_flag_get</code>
Function prototype		<code>FlagStatus rtc_flag_get(uint32_t flag);</code>
Function descriptions		get RTC flag status
Precondition		-
The called functions		-
Input parameter{in}		
flag		specify which RTC flag status to get
<code>RTC_FLAG_SECOND</code>		second interrupt flag
<code>RTC_FLAG_ALARM</code>		alarm interrupt flag
<code>RTC_FLAG_OVERFLOW</code>		overflow interrupt flag
<code>RTC_FLAG_RSYN</code>		registers synchronized flag
<code>RTC_FLAG_LWOF</code>		last write operation finished flag
Output parameter{out}		
-		-
Return value		
FlagStatus	SET or RESET	

Example:

```

/* get the RTC overflow interrupt 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 3-785. 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	-
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>	overflow interrupt flag
<i>RTC_FLAG_RSYN</i>	registers synchronized flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* clear the RTC alarm flag */

rtc_flag_clear (RTC_FLAG_ALARM);
  
```

3.23. SDIO

The secure digital input/output interface (SDIO) defines the SD/SD I/O /MMC CE-ATA card host interface, which provides command/data transfer between the 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 3-786. SDIO Registers

Registers	Descriptions
SDIO_PWRCTL	Power control register
SDIO_CLKCTL	Clock control register
SDIO_CMDAGMT	Command argument register
SDIO_CMDCTL	Command control register
SDIO_RSPCMDIDX	Command index response register
SDIO_RESPx x=0..3	Response register
SDIO_DATATO	Data timeout register
SDIO_DATALEN	Data length register
SDIO_DATACTL	Data control register
SDIO_DATACNT	Data counter register
SDIO_STAT	Status register
SDIO_INTC	Interrupt clear register
SDIO_INTEN	Interrupt enable register
SDIO_FIFOCNT	FIFO counter register
SDIO_FIFO	FIFO data register

3.23.2. Descriptions of Peripheral functions

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

Table 3-787. 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

Function name	Function description
sdio_dsm_disable	disable the DSM(data state machine)
sdio_data_write	write data(one word) to the transmit FIFO
sdio_data_read	read data(one word) from the receive FIFO
sdio_data_counter_get	get the number of remaining data bytes to be transferred to card
sdio_fifo_counter_get	get the number of words remaining to be written or read from FIFO
sdio_dma_enable	enable the DMA request for SDIO
sdio_dma_disable	disable the DMA request for SDIO
sdio_flag_get	get the flags state of SDIO
sdio_flag_clear	clear the pending flags of SDIO
sdio_interrupt_enable	enable the SDIO interrupt
sdio_interrupt_disable	disable the SDIO interrupt
sdio_interrupt_flag_get	get the interrupt flags state of SDIO
sdio_interrupt_flag_clear	clear the interrupt pending flags of SDIO
sdio_readwait_enable	enable the read wait mode(SD I/O only)
sdio_readwait_disable	disable the read wait mode(SD I/O only)
sdio_stop_readwait_enable	enable the function that stop the read wait process(SD I/O only)
sdio_stop_readwait_disable	disable the function that stop the read wait process(SD I/O only)
sdio_readwait_type_set	set the read wait type(SD I/O only)
sdio_operation_enable	enable the SD I/O mode specific operation(SD I/O only)
sdio_operation_disable	disable the SD I/O mode specific operation(SD I/O only)
sdio_suspend_enable	enable the SD I/O suspend operation(SD I/O only)
sdio_suspend_disable	disable the SD I/O suspend operation(SD I/O only)
sdio_ceata_command_enable	enable the CE-ATA command(CE-ATA only)
sdio_ceata_command_disable	disable the CE-ATA command(CE-ATA only)
sdio_ceata_interrupt_enable	enable the CE-ATA interrupt(CE-ATA only)
sdio_ceata_interrupt_disable	disable the CE-ATA interrupt(CE-ATA only)
sdio_ceata_command_completion_enable	enable the CE-ATA command completion signal(CE-ATA only)
sdio_ceata_command_completion_disable	disable the CE-ATA command completion signal(CE-ATA only)

sdio_deinit

The description of `sdio_deinit` is shown as below:

Table 3-788. Function `sdio_deinit`

Function name	<code>sdio_deinit</code>
Function prototype	<code>void sdio_deinit(void);</code>
Function descriptions	deinitialize the SDIO

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

Example:

```
/* deinitialize the SDIO */
sdio_deinit();
```

sdio_clock_config

The description of `sdio_clock_config` is shown as below:

Table 3-789. 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
<i>SDIO_SDIOCLKEDGE_RISING</i>	select the rising edge of the SDIOCLK to generate SDIO_CLK
<i>SDIO_SDIOCLKEDGE_FALLING</i>	select the falling edge of the SDIOCLK to generate SDIO_CLK
Input parameter{in}	
clock_bypass	clock bypass
<i>SDIO_CLOCKBYPASS_ENABLE</i>	clock bypass
<i>SDIO_CLOCKBYPASS_DISABLE</i>	no bypass
Input parameter{in}	
clock_powersave	SDIO_CLK clock dynamic switch on/off for power saving
<i>SDIO_CLOCKPWRSA_VE_ENABLE</i>	SDIO_CLK closed when bus is idle
<i>SDIO_CLOCKPWRSA_VE_DISABLE</i>	SDIO_CLK clock is always on
Input parameter{in}	

clock_division	clock division, less than 512
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the SDIO clock */

sdio_clock_config(SDIO_SDIOCLKEDGE_RISING,      SDIO_CLOCKBYPASS_DISABLE,
SDIO_CLOCKPWRSAVE_DISABLE, SD_CLK_DIV_TRANS);
```

sdio_hardware_clock_enable

The description of `sdio_hardware_clock_enable` is shown as below:

Table 3-790. 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 3-791. 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 3-792. Function `sdio_bus_mode_set`

Function name	sdio_bus_mode_set
Function prototype	void sdio_bus_mode_set(uint32_t bus_mode);
Function descriptions	set different SDIO card bus mode
Precondition	-
The called functions	-
Input parameter{in}	
bus_mode	SDIO card bus mode
<i>SDIO_BUSMODE_1BI</i> <i>T</i>	1-bit SDIO card bus mode
<i>SDIO_BUSMODE_4BI</i> <i>T</i>	4-bit SDIO card bus mode
<i>SDIO_BUSMODE_8BI</i> <i>T</i>	8-bit SDIO card bus mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set SDIO bus mode */

sdio_bus_mode_set(SDIO_BUSMODE_1BIT);
```

sdio_power_state_set

The description of `sdio_power_state_set` is shown as below:

Table 3-793. 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 3-794. 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 3-795. 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 3-796. 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 3-797. Function `sdio_command_response_config`

Function name	sdio_command_response_config
Function prototype	void sdio_command_response_config(uint32_t cmd_index, uint32_t cmd_argument, uint32_t response_type);

Function descriptions	configure the command and response
Precondition	-
The called functions	-
Input parameter{in}	
cmd_index	command index, refer to the related specifications
Input parameter{in}	
cmd_argument	command argument, refer to the related specifications
Input parameter{in}	
response_type	response type
<i>SDIO_RESPONSETYP_E_NO</i>	no response
<i>SDIO_RESPONSETYP_E_SHORT</i>	short response
<i>SDIO_RESPONSETYP_E_LONG</i>	long response
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CMD2(SD_CMD_ALL_SEND_CID) command response config*/
sdio_command_response_config(SD_CMD_ALL_SEND_CID, (uint32_t)0x0,
SDIO_RESPONSETYPE_LONG);
```

sdio_wait_type_set

The description of `sdio_wait_type_set` is shown as below:

Table 3-798. Function `sdio_wait_type_set`

Function name	sdio_wait_type_set
Function prototype	void sdio_wait_type_set(uint32_t wait_type);
Function descriptions	set the command state machine wait type
Precondition	-
The called functions	-
Input parameter{in}	
wait_type	wait type
<i>SDIO_WAITTYPE_NO</i>	not wait interrupt
<i>SDIO_WAITTYPE_INT_ERRUPT</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 3-799. 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 3-800. 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 3-801. Function sdio_command_index_get

Function name	sdio_command_index_get
Function prototype	uint8_t sdio_command_index_get(void);
Function descriptions	get the last response command index
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint8_t	last response command index

Example:

```
/* get SDIO command index */

uint8_t sdio_command_value;

sdio_command_value = sdio_command_index_get();
```

sdio_response_get

The description of sdio_response_get is shown as below:

Table 3-802. Function sdio_response_get

Function name	sdio_response_get
Function prototype	uint32_t sdio_response_get(uint32_t responsex);
Function descriptions	get the response for the last received command
Precondition	-
The called functions	-
Input parameter{in}	
responsex	SDIO response
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[0];

sdio_cid[0] = sdio_response_get(SDIO_RESPONSE0);
```

sdio_data_config

The description of `sdio_data_config` is shown as below:

Table 3-803. 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

<i>E_128BYTES</i>	
<i>SDIO_DATABLOCKSIZE_1BYTE</i>	block size = 1 byte
<i>E_256BYTES</i>	block size = 256 bytes
<i>SDIO_DATABLOCKSIZE_512BYTES</i>	block size = 512 bytes
<i>E_1024BYTES</i>	block size = 1024 bytes
<i>SDIO_DATABLOCKSIZE_2048BYTES</i>	block size = 2048 bytes
<i>E_4096BYTES</i>	block size = 4096 bytes
<i>SDIO_DATABLOCKSIZE_8192BYTES</i>	block size = 8192 bytes
<i>E_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 3-804. 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_TRANSDIRECTI ON_TOCARD</i>	write data to card
<i>SDIO_TRANSDIRECTI ON_TOSDIO</i>	read data from card
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SDIO data transmission */

sdio_data_transfer_config(SDIO_TRANSIRECTION_TOSDIO,
  SDIO_TRANSMODE_BLOCK);
```

sdio_dsm_enable

The description of `sdio_dsm_enable` is shown as below:

Table 3-805. Function `sdio_dsm_enable`

Function name	<code>sdio_dsm_enable</code>
Function prototype	<code>void sdio_dsm_enable(void);</code>
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 3-806. Function `sdio_dsm_disable`

Function name	<code>sdio_dsm_disable</code>
Function prototype	<code>void sdio_dsm_disable(void);</code>
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 3-807. 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 3-808. 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 3-809. Function `sdio_data_counter_get`

Function name	<code>sdio_data_counter_get</code>
Function prototype	<code>uint32_t sdio_data_counter_get(void);</code>
Function descriptions	get the number of remaining data bytes to be transferred to card
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	number of remaining data bytes to be transferred

Example:

```
/* get the number of remaining data bytes to be transferred to card */
uint32_t sdio_data_value;
sdio_data_value = sdio_data_counter_get();
```

sdio_fifo_counter_get

The description of `sdio_fifo_counter_get` is shown as below:

Table 3-810. Function `sdio_data_counter_get`

Function name	<code>sdio_fifo_counter_get</code>
Function prototype	<code>uint32_t sdio_fifo_counter_get(void);</code>
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 3-811. 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 3-812. Function `sdio_dma_disable`

Function name	sdio_dma_disable
Function prototype	void sdio_dma_disable(void);
Function descriptions	disable the DMA request for SDIO
Precondition	-
The called functions	-
Input parameter{in}	

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

Example:

```
/* disable the SDIO DMA */

sdio_dma_disable();
```

sdio_flag_get

The description of `sdio_flag_get` is shown as below:

Table 3-813. Function `sdio_flag_get`

Function name	<code>sdio_flag_get</code>
Function prototype	<code>FlagStatus sdio_flag_get(uint32_t flag);</code>
Function descriptions	get the flags state of SDIO
Precondition	-
The called functions	-
Input parameter{in}	
flag	flags state of SDIO
<i>SDIO_FLAG_CCRCER</i> <i>R</i>	command response received (CRC check failed) flag
<i>SDIO_FLAG_DTCRCE</i> <i>RR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_FLAG_CMDTMO</i> <i>UT</i>	command response timeout flag
<i>SDIO_FLAG_DTTMOU</i> <i>T</i>	data timeout flag
<i>SDIO_FLAG_TXURE</i>	transmit FIFO underrun error occurs flag
<i>SDIO_FLAG_RXORE</i>	received FIFO overrun error occurs flag
<i>SDIO_FLAG_CMDREC</i> <i>V</i>	command response received (CRC check passed) flag
<i>SDIO_FLAG_CMDSEN</i> <i>D</i>	command sent (no response required) flag
<i>SDIO_FLAG_DTEND</i>	data end (data counter, SDIO_DATACNT, is zero) flag
<i>SDIO_FLAG_STBITE</i>	start bit error in the bus flag
<i>SDIO_FLAG_DTBLKE</i> <i>ND</i>	data block sent/received (CRC check passed) flag
<i>SDIO_FLAG_CMDRUN</i>	command transmission in progress flag
<i>SDIO_FLAG_TXRUN</i>	data transmission in progress flag
<i>SDIO_FLAG_RXRUN</i>	data reception in progress flag

<i>SDIO_FLAG_TFH</i>	transmit FIFO is half empty flag: at least 8 words can be written into the FIFO
<i>SDIO_FLAG_RFH</i>	receive FIFO is half full flag: at least 8 words can be read in the FIFO
<i>SDIO_FLAG_TFF</i>	transmit FIFO is full flag
<i>SDIO_FLAG_RFF</i>	receive FIFO is full flag
<i>SDIO_FLAG_TFE</i>	transmit FIFO is empty flag
<i>SDIO_FLAG_RFE</i>	receive FIFO is empty flag
<i>SDIO_FLAG_TXDTVAL</i>	data is valid in transmit FIFO flag
<i>SDIO_FLAG_RXDTVAL</i>	data is valid in receive FIFO flag
<i>SDIO_FLAG_SDIOINT</i>	SD I/O interrupt received flag
<i>SDIO_FLAG_ATAEND</i>	CE-ATA command completion signal received (only for CMD61) flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the flags state of SDIO */

FlagStatus flag_value;

flag_value = sdio_flag_get(SDIO_FLAG_RFH);
```

sdio_flag_clear

The description of `sdio_flag_clear` is shown as below:

Table 3-814. Function `sdio_flag_clear`

Function name	<code>sdio_flag_clear</code>
Function prototype	<code>void sdio_flag_clear(uint32_t flag);</code>
Function descriptions	clear the pending flags of SDIO
Precondition	-
The called functions	-
Input parameter{in}	
flag	flags state of SDIO
<i>SDIO_FLAG_CCRCER</i> <i>R</i>	command response received (CRC check failed) flag
<i>SDIO_FLAG_DTCRCE</i> <i>RR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_FLAG_CMDTMO</i> <i>UT</i>	command response timeout flag
<i>SDIO_FLAG_DTTMOU</i> <i>T</i>	data timeout flag
<i>SDIO_FLAG_TXURE</i>	transmit FIFO underrun error occurs flag

<i>SDIO_FLAG_RXORE</i>	received FIFO overrun error occurs flag
<i>SDIO_FLAG_CMDREC</i> V	command response received (CRC check passed) flag
<i>SDIO_FLAG_CMDSEN</i> D	command sent (no response required) flag
<i>SDIO_FLAG_DTEND</i>	data end (data counter, SDIO_DATACNT, is zero) flag
<i>SDIO_FLAG_STBITE</i>	start bit error in the bus flag
<i>SDIO_FLAG_DTBLKE</i> ND	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 3-815. Function `sdio_interrupt_enable`

Function name	<code>sdio_interrupt_enable</code>
Function prototype	<code>void sdio_interrupt_enable(uint32_t int_flag);</code>
Function descriptions	enable the SDIO interrupt
Precondition	-
The called functions	-
Input parameter{in}	
<i>int_flag</i>	interrupt flags state of SDIO
<i>SDIO_INT_CCRCERR</i>	SDIO CCRCERR interrupt
<i>SDIO_INT_DTCRCER</i> R	SDIO DTCRCERR interrupt
<i>SDIO_INT_CMDTMOU</i> T	SDIO CMDTMOU interrupt
<i>SDIO_INT_DTTMOUT</i>	SDIO DTTMOUT interrupt
<i>SDIO_INT_TXURE</i>	SDIO TXURE interrupt
<i>SDIO_INT_RXORE</i>	SDIO_RXORE
<i>SDIO_INT_CMDRECV</i>	SDIO CMDRECV interrupt
<i>SDIO_INT_CMDSEND</i>	SDIO CMDSEND interrupt
<i>SDIO_INT_DTEND</i>	SDIO DTEND interrupt

<code>SDIO_INT_STBITE</code>	SDIO STBITE interrupt
<code>SDIO_INT_DTBKEND</code>	SDIO DTBKEND interrupt
<code>SDIO_INT_CMDRUN</code>	SDIO CMDRUN interrupt
<code>SDIO_INT_TXRUN</code>	SDIO TXRUN interrupt
<code>SDIO_INT_RXRUN</code>	SDIO RXRUN interrupt
<code>SDIO_INT_TFH</code>	SDIO TFH interrupt
<code>SDIO_INT_RFH</code>	SDIO RFH interrupt
<code>SDIO_INT_TFF</code>	SDIO TFF interrupt
<code>SDIO_INT_RFF</code>	SDIO RFF interrupt
<code>SDIO_INT_TFE</code>	SDIO TFE interrupt
<code>SDIO_INT_RFE</code>	SDIO RFE interrupt
<code>SDIO_INT_TXDTVAL</code>	SDIO TXDTVAL interrupt
<code>SDIO_INT_RXDTVAL</code>	SDIO RXDTVAL interrupt
<code>SDIO_INT_SDIOINT</code>	SDIO SDIOINT interrupt
<code>SDIO_INT_ATAEND</code>	SDIO ATAEND interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the SDIO corresponding interrupts */
sdio_interrupt_enable(SDIO_INT_CCRCERR | SDIO_INT_DTTMOUT | SDIO_INT_RXORE
| SDIO_INT_DTEND | SDIO_INT_STBITE);
```

sdio_interrupt_disable

The description of `sdio_interrupt_disable` is shown as below:

Table 3-816. Function `sdio_interrupt_disable`

Function name	<code>sdio_interrupt_disable</code>
Function prototype	<code>void sdio_interrupt_disable(uint32_t int_flag);</code>
Function descriptions	disable the SDIO interrupt
Precondition	-
The called functions	-
Input parameter{in}	
<code>int_flag</code>	interrupt flags state of SDIO
<code>SDIO_INT_CCRCERR</code>	SDIO CCRCERR interrupt
<code>SDIO_INT_DTCRCER</code> <i>R</i>	SDIO DTCRCERR interrupt
<code>SDIO_INT_CMDTMOU</code> <i>T</i>	SDIO CMDTMOU interrupt
<code>SDIO_INT_DTTMOUT</code>	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 3-817. 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
<i>SDIO_INT_FLAG_CCR CERR</i>	SDIO CCRCERR interrupt flag

<i>SDIO_INT_FLAG_DTC RCERR</i>	SDIO DTCRCERR interrupt flag
<i>SDIO_INT_FLAG_CMD TMOUT</i>	SDIO CMDTMOUT interrupt flag
<i>SDIO_INT_FLAG_DTT MOUT</i>	SDIO DTTMOUT interrupt flag
<i>SDIO_INT_FLAG_TXU RE</i>	SDIO TXURE interrupt flag
<i>SDIO_INT_FLAG_RXO RE</i>	SDIO_INT_RXORE flag
<i>SDIO_INT_FLAG_CMD RECV</i>	SDIO CMDRECV interrupt flag
<i>SDIO_INT_FLAG_CMD SEND</i>	SDIO CMDSEND interrupt flag
<i>SDIO_INT_FLAG_DTE ND</i>	SDIO DTEND interrupt flag
<i>SDIO_INT_FLAG_STB/ TE</i>	SDIO STBITE interrupt flag
<i>SDIO_INT_FLAG_DTB/ LKEND</i>	SDIO DTBLKEND interrupt flag
<i>SDIO_INT_FLAG_CMD/ RUN</i>	SDIO CMDRUN interrupt flag
<i>SDIO_INT_FLAG_TXR/ UN</i>	SDIO TXRUN interrupt flag
<i>SDIO_INT_FLAG_RXR/ UN</i>	SDIO RXRUN interrupt flag
<i>SDIO_INT_FLAG_TFH</i>	SDIO TFH interrupt flag
<i>SDIO_INT_FLAG_RFH</i>	SDIO RFH interrupt flag
<i>SDIO_INT_FLAG_TFF</i>	SDIO TFF interrupt flag
<i>SDIO_INT_FLAG_RFF</i>	SDIO RFF interrupt flag
<i>SDIO_INT_FLAG_TFE</i>	SDIO TFE interrupt flag
<i>SDIO_INT_FLAG_RFE</i>	SDIO RFE interrupt flag
<i>SDIO_INT_FLAG_TXD/ TVAL</i>	SDIO TXDTVAL interrupt flag
<i>SDIO_INT_FLAG_RXD/ TVAL</i>	SDIO RXDTVAL interrupt flag
<i>SDIO_INT_FLAG_SDI/ OINT</i>	SDIO SDIOINT interrupt flag
<i>SDIO_INT_FLAG_ATA/ END</i>	SDIO ATAEND interrupt flag
Output parameter{out}	
-	-
Return value	

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

Example:

```
/* get the interrupt flags state of SDIO */
FlagStatus flag_value;
flag_value = sdio_interrupt_flag_get(SDIO_INT_FLAG_DTEND);
```

sdio_interrupt_flag_clear

The description of `sdio_interrupt_flag_clear` is shown as below:

Table 3-818. 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</i> <i>CERR</i>	command response received (CRC check failed) flag
<i>SDIO_INT_FLAG_DTC</i> <i>RCERR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_INT_FLAG_CMD</i> <i>TMOUT</i>	command response timeout flag
<i>SDIO_INT_FLAG_DTT</i> <i>MOUT</i>	data timeout flag
<i>SDIO_INT_FLAG_TXU</i> <i>RE</i>	transmit FIFO underrun error occurs flag
<i>SDIO_INT_FLAG_RXO</i> <i>RE</i>	received FIFO overrun error occurs flag
<i>SDIO_INT_FLAG_CMD</i> <i>RECV</i>	command response received (CRC check passed) flag
<i>SDIO_INT_FLAG_CMD</i> <i>SEND</i>	command sent (no response required) flag
<i>SDIO_INT_FLAG_DTE</i> <i>ND</i>	data end (data counter, SDIO_DATACNT, is zero) flag
<i>SDIO_INT_FLAG_STBI</i> <i>TE</i>	start bit error in the bus flag
<i>SDIO_INT_FLAG_DTB</i> <i>LKEND</i>	data block sent/received (CRC check passed) flag
<i>SDIO_INT_FLAG_SDI</i> <i>OINT</i>	SD I/O interrupt received flag

SDIO_INT_FLAG_ATA END	CE-ATA command completion signal received (only for CMD61) flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the interrupt pending flags of SDIO */
sdio_interrupt_flag_clear(SDIO_INT_FLAG_DTEND);
```

sdio_readwait_enable

The description of `sdio_readwait_enable` is shown as below:

Table 3-819. Function `sdio_readwait_enable`

Function name	<code>sdio_readwait_enable</code>
Function prototype	<code>void sdio_readwait_enable(void);</code>
Function descriptions	enable the read wait mode(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the read wait mode(SD I/O only) */
sdio_readwait_enable();
```

sdio_readwait_disable

The description of `sdio_readwait_disable` is shown as below:

Table 3-820. Function `sdio_readwait_disable`

Function name	<code>sdio_readwait_disable</code>
Function prototype	<code>void sdio_readwait_disable(void);</code>
Function descriptions	disable the read wait mode(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the read wait mode(SD I/O only) */

sdio_readwait_disable();
```

sdio_stop_readwait_enable

The description of `sdio_stop_readwait_enable` is shown as below:

Table 3-821. 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 3-822. 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 3-823. 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_READWAITTYP_E_CLK	read wait control by stopping SDIO_CLK
SDIO_READWAITTYP_E_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(uint32_t readwait_type);
```

sdio_operation_enable

The description of `sdio_operation_enable` is shown as below:

Table 3-824. 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 3-825. 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 3-826. 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 3-827. 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 3-828. 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 3-829. 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 3-830. 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 3-831. 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 3-832. 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 3-833. Function `sdio_ceata_command_completion_disable`

Function name	sdio_ceata_command_completion_disable
Function prototype	void sdio_ceata_command_completion_disable(void);
Function descriptions	disable the CE-ATA command completion signal(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CE-ATA command completion signal(CE-ATA only) */

sdio_ceata_command_completion_disable();
```

3.24. SPI

The SPI/I2S module can communicate with external devices using the SPI protocol or the I2S audio protocol. The SPI/I2S registers are listed in chapter [3.24.1](#), the SPI/I2S firmware functions are introduced in chapter [3.24.2](#).

3.24.1. Descriptions of Peripheral registers

SPI/I2S registers are listed in the table shown as below:

Table 3-834. 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

Registers	Descriptions
SPI_I2SPSC	SPI/I2S clock prescaler register
SPI_QCTL	SPI quad mode control register

3.24.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

Table 3-835. SPI/I2S firmware function

Function name	Function description
spi_i2s_deinit	reset SPI and I2S peripheral
spi_struct_para_init	initialize the parameters of SPI struct with the default values
spi_init	initialize SPI peripheral parameter
spi_enable	enable SPI
spi_disable	disable SPI
i2s_init	initialize I2S peripheral parameter
i2s_psc_config	configure I2S peripheral prescaler
i2s_enable	enable I2S
i2s_disable	disable I2S
spi_nss_output_enable	enable SPI NSS output function
spi_nss_output_disable	disable SPI NSS output function
spi_nss_internal_high	SPI NSS pin high level in software mode
spi_nss_internal_low	SPI NSS pin low level in software mode
spi_dma_enable	enable SPI DMA function
spi_dma_disable	disable SPI DMA function
spi_i2s_data_frame_format_config	configure SPI/I2S data frame format
spi_i2s_data_transmit	SPI transmit data
spi_i2s_data_receive	SPI receive data
spi_bidirectional_transfer_config	configure SPI bidirectional transfer direction
spi_i2s_format_error_clear	clear TI Mode Format Error flag status
spi_crc_polynomial_set	set SPI CRC polynomial
spi_crc_polynomial_get	get SPI CRC polynomial
spi_crc_on	turn on SPI CRC function
spi_crc_off	turn off SPI CRC function
spi_crc_next	SPI next data is CRC value
spi_crc_get	get SPI CRC send value or receive value
spi_crc_error_clear	clear SPI CRC error flag status
spi_ti_mode_enable	enable SPI TI mode
spi_ti_mode_disable	disable SPI TI mode
spi_nssp_mode_enable	enable SPI NSS pulse mode
spi_nssp_mode_disable	disable SPI NSS pulse mode
i2s_full_duplex_mode_config	configure i2s full duplex mode
spi_quad_enable	enable SPI quad wire mode

Function name	Function description
spi_quad_disable	disable SPI quad wire mode
spi_quad_write_enable	enable SPI quad wire mode write
spi_quad_read_enable	enable SPI quad wire mode read
spi_quad_io23_output_enable	enable SPI quad wire mode SPI_IO2 and SPI_IO3 pin output
spi_quad_io23_output_disable	disable SPI quad wire mode SPI_IO2 and SPI_IO3 pin output
spi_i2s_interrupt_enable	enable SPI and I2S interrupt
spi_i2s_interrupt_disable	disable SPI and I2S interrupt
spi_i2s_interrupt_flag_get	get SPI and I2S interrupt status
spi_i2s_flag_get	get SPI and I2S flag status

Structure spi_parameter_struct

Table 3-836. spi_parameter_struct

Member name	Function description
device_mode	SPI master or slave (SPI_MASTER, SPI_SLAVE)
trans_mode	SPI transtype (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	SPI frame size (SPI_FRAMESIZE_16BIT, SPI_FRAMESIZE_8BIT)
nss	SPI NSS control by hardware or software (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	SPI big endian or little endian (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	SPI clock phase and polarity (SPI_CK_PL_LOW_PH_1EDGE, SPI_CK_PL_HIGH_PH_1EDGE, SPI_CK_PL_LOW_PH_2EDGE, SPI_CK_PL_HIGH_PH_2EDGE)
prescale	SPI prescale factor (SPI_PSC_n (n=2,4,8,16,32,64,128,256))

spi_i2s_deinit

The description of spi_i2s_deinit is shown as below:

Table 3-837. Function spi_i2s_deinit

Function name	spi_i2s_deinit
Function prototype	void spi_i2s_deinit(uint32_t spi_periph);
Function descriptions	reset SPI and I2S peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
spi_periph	SPI/I2S peripheral

SPIx	x=0,1,2
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 3-838. Function spi_struct_para_init

Function name	spi_struct_para_init
Function prototype	void spi_struct_para_init(spi_parameter_struct* spi_struct);
Function descriptions	initialize the parameters of SPI struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
*spi_struct	a spi_parameter_struct address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the parameters of SPI */
spi_parameter_struct spi_init_struct;
spi_struct_para_init(&spi_init_struct);
```

spi_init

The description of spi_init is shown as below:

Table 3-839. Function spi_init

Function name	spi_init
Function prototype	void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
Function descriptions	initialize SPI peripheral parameter
Precondition	-
The called functions	-
Input parameter{in}	

spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2
Input parameter{in}	
spi_struct	SPI parameter initialization stuct, the structure members can refer to members of the structure Table 3-836. spi_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize SPI0 */

spi_parameter_struct spi_init_struct;

spi_init_struct.trans_mode          = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode         = SPI_MASTER;
spi_init_struct.frame_size         = SPI_FRAMESIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss                = SPI_NSS_SOFT;
spi_init_struct.prescale           = SPI_PSC_8;
spi_init_struct.endian              = SPI_ENDIAN_MSB;

spi_init(SPI0, &spi_init_struct);
```

spi_enable

The description of `spi_enable` is shown as below:

Table 3-840. 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
<i>SPIx</i>	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 */
spi_enable(SPI0);
```

spi_disable

The description of spi_disable is shown as below:

Table 3-841. Function spi_disable

Function name	spi_disable
Function prototype	void spi_disable(uint32_t spi_periph);
Function descriptions	disable SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 */
spi_disable(SPI0);
```

i2s_init

The description of i2s_init is shown as below:

Table 3-842. Function i2s_init

Function name	i2s_init
Function prototype	void i2s_init(uint32_t spi_periph,uint32_t i2s_mode, uint32_t i2s_standard, uint32_t i2s_ckpl);
Function descriptions	initialize I2S peripheral parameter
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	I2S peripheral
SPIx	x=1,2
Input parameter{in}	
i2s_mode	I2S operation mode
I2S_MODE_SLAVE_TX	I2S slave transmit mode

<i>I2S_MODE_SLAVE_RX</i>	I2S slave receive mode
<i>I2S_MODE_MASTER_TX</i>	I2S master transmit mode
<i>I2S_MODE_MASTER_RX</i>	I2S master receive mode
Input parameter{in}	
<i>i2s_standard</i>	I2S standard
<i>I2S_STD_PHILLIPS</i>	I2S phillips standard
<i>I2S_STD_MSB</i>	I2S MSB standard
<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCM_SHORT</i>	I2S PCM short standard
<i>I2S_STD_PCM_LONG</i>	I2S PCM long standard
Input parameter{in}	
<i>i2s_ckpl</i>	I2S idle state clock polarity
<i>I2S_CKPL_LOW</i>	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_MASTER_TX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

i2s_psc_config

The description of i2s_psc_config is shown as below:

Table 3-843. Function i2s_psc_config

Function name	i2s_psc_config
Function prototype	void i2s_psc_config(uint32_t spi_periph, uint32_t i2s_audiosample, uint32_t i2s_frameformat, uint32_t i2s_mckout);
Function descriptions	configure I2S prescaler
Precondition	-
The called functions	rcu_clock_freq_get
Input parameter{in}	
<i>spi_periph</i>	I2S peripheral
<i>SPIx</i>	x=1,2
Input parameter{in}	
<i>i2s_audiosample</i>	I2S audio sample rate
<i>I2S_AUDIOSAMPLE_8K</i>	audio sample rate is 8KHz

<i>I2S_AUDIOSAMPLE_1</i> 1K	audio sample rate is 11KHz
<i>I2S_AUDIOSAMPLE_1</i> 6K	audio sample rate is 16KHz
<i>I2S_AUDIOSAMPLE_2</i> 2K	audio sample rate is 22KHz
<i>I2S_AUDIOSAMPLE_3</i> 2K	audio sample rate is 32KHz
<i>I2S_AUDIOSAMPLE_4</i> 4K	audio sample rate is 44KHz
<i>I2S_AUDIOSAMPLE_4</i> 8K	audio sample rate is 48KHz
<i>I2S_AUDIOSAMPLE_9</i> 6K	audio sample rate is 96KHz
<i>I2S_AUDIOSAMPLE_1</i> 92K	audio sample rate is 192KHz
Input parameter{in}	
<i>i2s_frameformat</i>	I2S data length and channel length
<i>I2S_FRAMEFORMAT_</i> <i>DT16B_CH16B</i>	I2S data length is 16 bit and channel length is 16 bit
<i>I2S_FRAMEFORMAT_</i> <i>DT16B_CH32B</i>	I2S data length is 16 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_</i> <i>DT24B_CH32B</i>	I2S data length is 24 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_</i> <i>DT32B_CH32B</i>	I2S data length is 32 bit and channel length is 32 bit
Input parameter{in}	
<i>i2s_mckout</i>	I2S master clock output
<i>I2S_MCKOUT_ENABL</i> <i>E</i>	I2S master clock output enable
<i>I2S_MCKOUT_DISABLE</i> <i>E</i>	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 3-844. Function i2s_enable

Function name	i2s_enable
Function prototype	void i2s_enable(uint32_t spi_periph);
Function descriptions	enable I2S
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	I2S peripheral
SPIx	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2S1*/
i2s_enable(SPI1);
```

i2s_disable

The description of i2s_disable is shown as below:

Table 3-845. Function i2s_disable

Function name	i2s_disable
Function prototype	void i2s_disable(uint32_t spi_periph);
Function descriptions	disable I2S
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	I2S peripheral
SPIx	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2S1*/
i2s_disable(SPI1);
```

spi_nss_output_enable

The description of spi_nss_output_enable is shown as below:

Table 3-846. Function spi_nss_output_enable

Function name	spi_nss_output_enable
Function prototype	void spi_nss_output_enable(uint32_t spi_periph);
Function descriptions	enable SPI NSS output function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
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 3-847. Function spi_nss_output_disable

Function name	spi_nss_output_disable
Function prototype	void spi_nss_output_disable(uint32_t spi_periph);
Function descriptions	disable SPI NSS output function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
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 3-848. Function spi_nss_internal_high

Function name	spi_nss_internal_high
Function prototype	void spi_nss_internal_high(uint32_t spi_periph);
Function descriptions	SPI NSS pin high level in software mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
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 3-849. Function spi_nss_internal_low

Function name	spi_nss_internal_low
Function prototype	void spi_nss_internal_low(uint32_t spi_periph);
Function descriptions	SPI NSS pin low level in software mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
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 3-850. Function spi_dma_enable

Function name	spi_dma_enable
Function prototype	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
Function descriptions	enable SPI DMA function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Input parameter{in}	
dma	SPI DMA mode
SPI_DMA_TRANSMIT	SPI transmit data use DMA
SPI_DMA_RECEIVE	SPI receive data use DMA
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 transmit data DMA function */
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

spi_dma_disable

The description of spi_dma_disable is shown as below:

Table 3-851. Function spi_dma_disable

Function name	spi_dma_disable
Function prototype	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
Function descriptions	disable SPI DMA function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Input parameter{in}	
dma	SPI DMA mode
SPI_DMA_TRANSMIT	SPI transmit data use DMA
SPI_DMA_RECEIVE	SPI receive data use DMA
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable SPI0 transmit data DMA function */
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

spi_i2s_data_frame_format_config

The description of `spi_i2s_data_frame_format_config` is shown as below:

Table 3-852. Function `spi_i2s_data_frame_format_config`

Function name	spi_i2s_data_frame_format_config
Function prototype	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
Function descriptions	configure SPI/I2S data frame format
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Input parameter{in}	
frame_format	SPI frame size
SPI_FRAMESIZE_16BIT	SPI frame size is 16 bits
SPI_FRAMESIZE_8BIT	SPI frame size is 8 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SPI1/I2S1 data frame format size is 16 bits */
spi_i2s_data_frame_format_config(SPI1, SPI_FRAMESIZE_16BIT);
```

spi_i2s_data_transmit

The description of `spi_i2s_data_transmit` is shown as below:

Table 3-853. Function `spi_i2s_data_transmit`

Function name	spi_i2s_data_transmit
Function prototype	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
Function descriptions	SPI transmit data

Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Input parameter{in}	
data	16-bit data
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 transmit data */

uint16_t spi0_send_array[] = {0x5050,0XA0A0};

spi_i2s_data_transmit(SPI0, spi0_send_array[0]);
```

spi_i2s_data_receive

The description of spi_i2s_data_receive is shown as below:

Table 3-854. 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
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
uint16_t	16-bit data

Example:

```
/* SPI0 receive data */

uint16_t spi0_receive_data;

spi0_receive_data = spi_i2s_data_receive(SPI0);
```

spi_bidirectional_transfer_config

The description of spi_bidirectional_transfer_config is shown as below:

Table 3-855. 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
SPIx	x=0,1,2
Input parameter{in}	
transfer_direction	SPI transfer direction
SPI_BIDIRECTIONAL_TRANSMIT	SPI work in transmit-only mode
SPI_BIDIRECTIONAL_RECEIVE	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_format_error_clear

The description of spi_i2s_format_error_clear is shown as below:

Table 3-856. Function spi_i2s_format_error_clear

Function name	spi_i2s_format_error_clear
Function prototype	void spi_i2s_format_error_clear(uint32_t spi_periph, uint32_t flag);
Function descriptions	clear TI Mode Format Error flag status
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Input parameter{in}	
flag	SPI/I2S format error flag

<i>SPI_FLAG_FERR</i>	Format error only for SPI work in TI mode
<i>I2S_FLAG_FERR</i>	Format error for I2S
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear SPI0 format error flag status */

spi_i2s_format_error_clear (SPI0, SPI_FLAG_FERR);
```

spi_crc_polynomial_set

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

Table 3-857. Function spi_crc_polynomial_set

Function name	spi_crc_polynomial_set
Function prototype	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);
Function descriptions	set SPI CRC polynomial
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Input parameter{in}	
crc_poly	CRC polynomial value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set SPI0 CRC polynomial */

spi_crc_polynomial_set(SPI0,CRC_VALUE);
```

spi_crc_polynomial_get

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

Table 3-858. Function spi_crc_polynomial_get

Function name	spi_crc_polynomial_get
Function prototype	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
Function descriptions	get SPI CRC polynomial
Precondition	-

The called functions		-
Input parameter{in}		
spi_periph		SPI peripheral
SPIx		x=0,1,2
Output parameter{out}		
-		-
Return value		
uint16_t		16-bit CRC polynomial (0-0xFFFF)

Example:

```
/* get SPI0 CRC polynomial */

uint16_t crc_val;

crc_val = spi_crc_polynomial_get(SPI0);
```

spi_crc_on

The description of spi_crc_on is shown as below:

Table 3-859. Function spi_crc_on

Function name	spi_crc_on
Function prototype	void spi_crc_on(uint32_t spi_periph);
Function descriptions	turn on SPI CRC function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
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 3-860. Function spi_crc_off

Function name	spi_crc_off
Function prototype	void spi_crc_off(uint32_t spi_periph);

Function descriptions	turn off SPI CRC function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
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 3-861. Function spi_crc_next

Function name	spi_crc_next
Function prototype	void spi_crc_next(uint32_t spi_periph);
Function descriptions	SPI next data is CRC value
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 next data is CRC value */

spi_crc_next(SPI0);
```

spi_crc_get

The description of spi_crc_get is shown as below:

Table 3-862. Function spi_crc_get

Function name	spi_crc_get
Function prototype	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc);

Function descriptions	get SPI CRC send value or receive value
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Input parameter{in}	
crc	SPI crc value
SPI_CRC_TX	get transmit crc value
SPI_CRC_RX	get receive crc value
Output parameter{out}	
-	-
Return value	
uint16_t	16-bit CRC value (0-0xFFFF)

Example:

```
/* get SPI0 CRC send value */

uint16_t crc_val;

crc_val = spi_crc_get(SPI0, SPI_CRC_TX);
```

spi_crc_error_clear

The description of `spi_crc_error_clear` is shown as below:

Table 3-863. Function `spi_crc_error_clear`

Function name	spi_crc_error_clear
Function prototype	void spi_crc_error_clear(uint32_t spi_periph);
Function descriptions	clear SPI CRC error flag status
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear SPI0 CRC error flag status */

spi_crc_error_clear(SPI0);
```

spi_ti_mode_enable

The description of spi_ti_mode_enable is shown as below:

Table 3-864. Function spi_ti_mode_enable

Function name	spi_ti_mode_enable
Function prototype	void spi_ti_mode_enable(uint32_t spi_periph);
Function descriptions	enable SPI TI mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 TI mode */
spi_ti_mode_enable(SPI0);
```

spi_ti_mode_disable

The description of spi_ti_mode_disable is shown as below:

Table 3-865. Function spi_ti_mode_disable

Function name	spi_ti_mode_disable
Function prototype	void spi_ti_mode_disable(uint32_t spi_periph);
Function descriptions	disable SPI TI mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

spi_nssp_mode_enable

The description of spi_nssp_mode_enable is shown as below:

Table 3-866. Function spi_nssp_mode_enable

Function name	spi_nssp_mode_enable
Function prototype	void spi_nssp_mode_enable(uint32_t spi_periph);
Function descriptions	enable SPI NSS pulse mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 NSS pulse mode */

spi_nssp_mode_enable(SPI0);
```

spi_nssp_mode_disable

The description of spi_nssp_mode_disable is shown as below:

Table 3-867. Function spi_nssp_mode_disable

Function name	spi_nssp_mode_disable
Function prototype	void spi_nssp_mode_disable(uint32_t spi_periph);
Function descriptions	disable SPI NSS pulse mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 NSS pulse mode */

spi_nssp_mode_disable(SPI0);
```

i2s_full_duplex_mode_config

The description of i2s_full_duplex_mode_config is shown as below:

Table 3-868. Function i2s_init

Function name	i2s_full_duplex_mode_config
Function prototype	void i2s_full_duplex_mode_config(uint32_t i2s_add_periph,uint32_t i2s_mode,uint32_t i2s_standard,uint32_t i2s_ckpl,uint32_t i2s_frameformat);
Function descriptions	configure i2s full duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
i2s_add_periph	I2S_ADD peripheral
I2Sx_ADD	x=1,2
Input parameter{in}	
i2s_mode	I2S operation mode
I2S_MODE_SLAVETX	I2S slave transmit mode
I2S_MODE_SLAVERX	I2S slave receive mode
I2S_MODE_MASTERTX	I2S master transmit mode
I2S_MODE_MASTERRX	I2S master receive mode
Input parameter{in}	
i2s_standard	I2S standard
I2S_STD_PHILLIPS	I2S phillips standard
I2S_STD_MSB	I2S MSB standard
I2S_STD_LSB	I2S LSB standard
I2S_STD_PCMSHORT	I2S PCM short standard
I2S_STD_PCMLONG	I2S PCM long standard
Input parameter{in}	
i2s_ckpl	I2S idle state clock polarity
I2S_CKPL_LOW	I2S clock polarity low level
I2S_CKPL_HIGH	I2S clock polarity high level
Input parameter{in}	
i2s_frameformat	I2S data length and channel length
I2S_FRAMEFORMAT_DT16B_CH16B	I2S data length is 16 bit and channel length is 16 bit
I2S_FRAMEFORMAT_DT16B_CH32B	I2S data length is 16 bit and channel length is 32 bit
I2S_FRAMEFORMAT_DT24B_CH32B	I2S data length is 24 bit and channel length is 32 bit
I2S_FRAMEFORMAT_	I2S data length is 32 bit and channel length is 32 bit

<i>DT32B_CH32B</i>	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure I2S1_ADD */

i2s_full_duplex_mode_config(I2S1_ADD,I2S_MODE_MASTERTX,I2S_STD_PHILLIPS,
I2S_CKPL_HIGH, I2S_FRAMEFORMAT_DT16B_CH16B);
```

spi_quad_enable

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

Table 3-869. Function spi_quad_enable

Function name	spi_quad_enable
Function prototype	void spi_quad_enable (uint32_t spi_periph);
Function descriptions	enable SPI quad wire mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 quad wire mode */

spi_quad_enable (SPI0);
```

spi_quad_disable

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

Table 3-870. Function spi_quad_disable

Function name	spi_quad_disable
Function prototype	spi_quad_disable (uint32_t spi_periph);
Function descriptions	disable SPI quad wire mode
Precondition	-
The called functions	-
Input parameter{in}	

spi_periph	SPI peripheral
<i>SPIx</i>	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 quad wire mode */

spi_quad_disable (SPI0);
```

spi_quad_write_enable

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

Table 3-871. 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 SPI quad wire mode write
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 quad wire write */

spi_quad_write_enable (SPI0);
```

spi_quad_read_enable

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

Table 3-872. 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 SPI quad wire mode read
Precondition	-
The called functions	-
Input parameter{in}	

spi_periph	SPI peripheral
<i>SPIx</i>	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 quad wire read */
spi_quad_read_enable (SPI0);
```

spi_quad_io23_output_enable

The description of **spi_quad_io23_output_enable** is shown as below:

Table 3-873. 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 quad wire mode 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:

```
/* enable SPI0 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 3-874. 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 quad wire mode 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 SPI0 SPI_IO2 and SPI_IO3 pin output */

spi_quad_io23_output_disable (SPI0);
```

spi_i2s_interrupt_enable

The description of **spi_i2s_interrupt_enable** is shown as below:

Table 3-875. 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 peripheral
<i>SPIx</i>	x=0,1,2
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 3-876. 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}	
<code>spi_periph</code>	SPI peripheral
<code>SPIx</code>	x=0,1,2
Input parameter{in}	
<code>interrupt</code>	SPI/I2S interrupt
<code>SPI_I2S_INT_TBE</code>	transmit buffer empty interrupt
<code>SPI_I2S_INT_RBNE</code>	receive buffer not empty interrupt
<code>SPI_I2S_INT_ERR</code>	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 3-877. 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}	
<code>spi_periph</code>	SPI peripheral
<code>SPIx</code>	x=0,1,2
Input parameter{in}	
<code>interrupt</code>	SPI/I2S interrupt flag status
<code>SPI_I2S_INT_FLAG_TBE</code>	transmit buffer empty interrupt
<code>SPI_I2S_INT_FLAG_RBN</code>	receive buffer not empty interrupt

<i>SPI_I2S_INT_FLAG_R</i> <i>XORERR</i>	overrun interrupt
<i>SPI_INT_FLAG_CONF</i> <i>ERR</i>	config error interrupt
<i>SPI_INT_FLAG_CRC</i> <i>RR</i>	CRC error interrupt
<i>I2S_INT_FLAG_TXUR</i> <i>ERR</i>	underrun error interrupt
<i>SPI_I2S_INT_FLAG_F</i> <i>ERR</i>	format error interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty interrupt status */

if(RESET != spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE)){
    while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
    spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
}
```

spi_i2s_flag_get

The description of spi_i2s_flag_get is shown as below:

Table 3-878. 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 peripheral
SPIx	x=0,1,2
Input parameter{in}	
flag	SPI/I2S flag status
SPI_FLAG_TBE	transmit buffer empty flag
SPI_FLAG_RBNE	receive buffer not empty flag
SPI_FLAG_TRANS	transmit on-going flag
SPI_I2S_INT_FLAG_R <i>XORERR</i>	receive overrun error flag

<i>SPI_FLAG_CONFERR</i>	mode config error flag
<i>SPI_FLAG_CRCERR</i>	CRC error flag
<i>SPI_FLAG_FREE</i>	format error flag
<i>I2S_FLAG_TBE</i>	transmit buffer empty flag
<i>I2S_FLAG_RBNE</i>	receive buffer not empty flag
<i>I2S_FLAG_TRANS</i>	transmit on-going flag
<i>I2S_FLAG_RXORERR</i>	overrun error flag
<i>I2S_FLAG_TXURERR</i>	underrun error flag
<i>I2S_FLAG_CH</i>	channel side flag
<i>I2S_FLAG_FERR</i>	format error flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty flag status */
while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

3.25. SQPI

Serial/Quad Parallel Interface (SQPI) is a controller for external serial/dual/quad parallel interface memory peripheral. The SQPI registers are listed in chapter [3.25.1](#), the SQPI firmware functions are introduced in chapter [3.25.2](#).

3.25.1. Descriptions of Peripheral registers

SQPI registers are listed in the table shown as below:

Table 3-879. SQPI Registers

Registers	Descriptions
SQPI_INIT	SQPI initial register
SQPI_RCMD	SQPI read command register
SQPI_WCMD	SQPI write command register
SQPI_IDL	SQPI ID low register
SQPI_IDH	SQPI ID high register

3.25.2. Descriptions of Peripheral functions

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

Table 3-880. SQPI firmware function

Function name	Function description
sqpi_deinit	reset SQPI peripheral
sqpi_struct_para_init	initialize the parameters of SQPI struct with the default values
sqpi_init	initialize SQPI peripheral parameter
sqpi_read_id_command	send SQPI read ID command
sqpi_special_command	send SQPI special command
sqpi_read_command_config	configure SQPI read command
sqpi_write_command_config	configure SQPI write command
sqpi_low_id_receive	SQPI receive low ID
sqpi_high_id_receive	SQPI receive high ID

Structure sqpi_parameter_struct

Table 3-881. sqpi_parameter_struct

Member name	Function description
polarity	SQPI sample polarity (SQPI_SAMPLE_POLARITY_RISING, SQPI_SAMPLE_POLARITY_FALLING)
id_length	external memory ID length (QSPI_ID_LENGTH_n_BITS (n=8,16,32,64))
addr_bit	bit number of SPI PSRAM address phase (0x00 - 0x1F)
clk_div	clock divider for SQPI output clock (0x01 - 0x3F)
cmd_bit	bit number of SQPI controller command phase (QSPI_CMDBIT_n_BITS (n=4,8,16))

sqpi_deinit

The description of sqpi_deinit is shown as below:

Table 3-882. Function sqpi_deinit

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

Example:

```
/* reset SQPI */
```

```
sqpi_deinit();
```

sqpi_struct_para_init

The description of sqpi_struct_para_init is shown as below:

Table 3-883. Function sqpi_struct_para_init

Function name	sqpi_struct_para_init	
Function prototype	void sqpi_struct_para_init(sqpi_parameter_struct* sqpi_struct);	
Function descriptions	initialize the parameters of SQPI struct with the default values	
Precondition	-	
The called functions	-	
Input parameter{in}		
*sqpi_struct	a sqpi_parameter_struct address	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

```
/* initialize the parameters of SPI */
sqpi_parameter_struct sqpi_init_struct;
sqpi_struct_para_init(&sqpi_init_struct);
```

sqpi_init

The description of sqpi_init is shown as below:

Table 3-884. Function sqpi_init

Function name	sqpi_init	
Function prototype	void sqpi_init(sqpi_parameter_struct* sqpi_struct);	
Function descriptions	initialize SQPI peripheral parameter	
Precondition	-	
The called functions	-	
Input parameter{in}		
spi_struct	SPI parameter initialization stuct, the structure members can refer to members of the structure Table 3-881. sqpi_parameter_struct	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

```

/* initialize SQPI */

sqpi_parameter_struct sqpi_init_struct;

sqpi_struct->polarity = SQPI_SAMPLE_POLARITY_RISING;

sqpi_struct->id_length = QSPI_ID_LENGTH_32_BITS;

sqpi_struct->addr_bit = 24U;

sqpi_struct->clk_div = 2U;

sqpi_struct->cmd_bit = QSPI_CMDBIT_8_BITS;

sqpi_init(&sqpi_init_struct);

```

sqpi_read_id_command

The description of `sqpi_read_id_command` is shown as below:

Table 3-885. Function `sqpi_read_id_command`

Function name	sqpi_read_id_command
Function prototype	void sqpi_read_id_command (void);
Function descriptions	send SQPI read ID command
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* send SQPI read ID command */

sqpi_read_id_command ();

```

sqpi_special_command

The description of `sqpi_special_command` is shown as below:

Table 3-886. Function `sqpi_special_command`

Function name	sqpi_special_command
Function prototype	void sqpi_special_command(void);
Function descriptions	send SQPI special command
Precondition	-
The called functions	-
Input parameter{in}	

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

Example:

```
/* send SQPI special command */

sqpi_special_command();
```

sqpi_read_command_config

The description of `sqpi_read_command_config` is shown as below:

Table 3-887. Function `sqpi_read_command_config`

Function name	<code>sqpi_read_command_config</code>
Function prototype	<code>void sqpi_read_command_config(uint32_t rmode, uint32_t rwaitcycle, uint32_t rcmd);</code>
Function descriptions	configure SQPI read command
Precondition	-
The called functions	-
Input parameter{in}	
rmode	SQPI read command mode
<code>QSPI_MODE_SSQ</code>	SSQ mode
<code>QSPI_MODE_SSS</code>	SSS mode
<code>QSPI_MODE_SQQ</code>	SQQ mode
<code>QSPI_MODE_QQQ</code>	QQQ mode
<code>QSPI_MODE_SSD</code>	SSD mode
<code>QSPI_MODE_SDD</code>	SDD mode
Input parameter{in}	
rwaitcycle	SQPI read wait cycle (0x00 – 0x1F)
Input parameter{in}	
rcmd	SQPI read command(0x00 – 0xFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SQPI read command */

sqpi_read_command_config(QSPI_MODE_SSS,0x00,0x9f);
```

sqpi_write_command_config

The description of sqpi_write_command_config is shown as below:

Table 3-888. Function sqpi_write_command_config

Function name	sqpi_write_command_config
Function prototype	void sqpi_write_command_config(uint32_t wmode, uint32_t wwaitcycle, uint32_t wcmd);
Function descriptions	configure SQPI write command
Precondition	-
The called functions	-
Input parameter{in}	
wmode	SQPI write command mode
<i>QSPI_MODE_SSQ</i>	SSQ mode
<i>QSPI_MODE_SSS</i>	SSS mode
<i>QSPI_MODE_SQQ</i>	SQQ mode
<i>QSPI_MODE_QQQ</i>	QQQ mode
<i>QSPI_MODE_SSD</i>	SSD mode
<i>QSPI_MODE_SDD</i>	SDD mode
Input parameter{in}	
wwaitcycle	SQPI write wait cycle (0x00 – 0x1F)
Input parameter{in}	
wcmd	SQPI write command(0x00 – 0xFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SQPI write command */
sqpi_write_command_config(QSPI_MODE_SSS,0x00,0x9f);
```

sqpi_low_id_receive

The description of sqpi_low_id_receive is shown as below:

Table 3-889. Function sqpi_low_id_receive

Function name	sqpi_low_id_receive
Function prototype	uint32_t sqpi_low_id_receive(void);
Function descriptions	get low ID value
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
uint32_t	32-bit IDvalue (0-0xFFFF)

Example:

```
/* get SQPI low ID value */

uint32_t val;

val = sqpi_low_id_receive();
```

sqpi_high_id_receive

The description of sqpi_high_id_receive is shown as below:

Table 3-890. Function sqpi_low_id_receive

Function name	sqpi_high_id_receive	
Function prototype	uint32_t sqpi_high_id_receive(void);	
Function descriptions	get high ID value	
Precondition	-	
The called functions	-	
Input parameter{in}		
-	-	
Output parameter{out}		
-	-	
Return value		
uint32_t	32-bit ID value (0-0xFFFF)	

Example:

```
/* get SQPI high ID value */

uint32_t val;

val = sqpi_high_id_receive();
```

3.26. TIMER

The timers have a 16-bit counter that can be used as an unsigned counter and supports both input capture and output compare. Timers (TIMERx) are divided into five sorts: advanced timer (TIMERx, x=0, 7), general level0 timer (TIMERx, x=1, 2, 3, 4), general level1 timer (TIMERx, x=8, 11), general level2 timer (TIMERx, x=9, 10, 12, 13), basic timer (TIMERx, x=5, 6). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.26.1](#), the TIMER firmware functions are introduced in chapter [3.26.2](#).

3.26.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

Table 3-891. TIMERx Registers

Registers	Descriptions
TIMER_CTL0	Control register 0
TIMER_CTL1	Control register 1
TIMER_SMCFG	Slave mode configuration register
TIMER_DMAMINTEN	DMA and interrupt enable register
TIMER_INTF	Interrupt flag register
TIMER_SWEVG	Software event generation register
TIMER_CHCTL0	Channel control register 0
TIMER_CHCTL1	Channel control register 1
TIMER_CHCTL2	Channel control register 2
TIMER_CNT	Counter register
TIMER_PSC	Prescaler register
TIMER_CAR	Counter auto reload register
TIMER_CREP	Counter repetition register
TIMER_CH0CV	Channel 0 capture/compare value register
TIMER_CH1CV	Channel 1 capture/compare value register
TIMER_CH2CV	Channel 2 capture/compare value register
TIMER_CH3CV	Channel 3 capture/compare value register
TIMER_CCHP	Channel complementary protection register
TIMER_DMACFG	DMA configuration register
TIMER_DMATB	DMA transfer buffer register
TIMER_CFG	Configuration register

3.26.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

Table 3-892. TIMERx firmware function

Function name	Function description
timer_deinit	deinit a timer
timer_struct_para_init	initialize the parameters of TIMER init parameter struct with the default values
timer_init	initialize TIMER counter
timer_enable	enable a timer
timer_disable	disable a timer
timer_auto_reload_shadow_enable	enable the auto reload shadow function
timer_auto_reload_shadow_disable	disable the auto reload shadow function
timer_update_event_enable	enable the update event

Function name	Function description
timer_update_event_disable	disable the update event
timer_counter_alignment	set TIMER counter alignment mode
timer_counter_up_direction	set TIMER counter up direction
timer_counter_down_direction	set TIMER counter down direction
timer_prescaler_config	configure TIMER prescaler
timer_repetition_value_config	configure TIMER repetition register value
timer_autoreload_value_config	configure TIMER autoreload register value
timer_counter_value_config	configure TIMER counter register value
timer_counter_read	read TIMER counter value
timer_prescaler_read	read TIMER prescaler value
timer_single_pulse_mode_config	configure TIMER single pulse mode
timer_update_source_config	configure TIMER update source
timer_dma_enable	enable the TIMER DMA
timer_dma_disable	disable the TIMER DMA
timer_channel_dma_request_source_select	channel DMA request source selection
timer_dma_transfer_config	configure the TIMER DMA transfer
timer_event_software_generate	software generate events
timer_break_struct_para_init	initialize the parameters of TIMER break parameter struct with the default values
timer_break_config	configure TIMER break function
timer_break_enable	enable TIMER break function
timer_break_disable	disable TIMER break function
timer_automatic_output_enable	enable TIMER output automatic function
timer_automatic_output_disable	disable TIMER output automatic function
timer_primary_output_config	configure TIMER primary output function
timer_channel_control_shadow_config	channel capture/compare control shadow register enable
timer_channel_control_shadow_update_config	configure TIMER channel control shadow register update control
timer_channel_output_struct_para_init	initialize the parameters of TIMER channel output parameter struct with the default values
timer_channel_output_config	configure TIMER channel output function
timer_channel_output_mode_config	configure TIMER channel output compare mode
timer_channel_output_pulse_value_config	configure TIMER channel output pulse value
timer_channel_output_shadow_config	configure TIMER channel output shadow function
timer_channel_output_fast_config	configure TIMER channel output fast function
timer_channel_output_clear_config	configure TIMER channel output clear function
timer_channel_output_polarity_config	configure TIMER channel output polarity

Function name	Function description
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 the parameters of TIMER channel input parameter struct with the default values
timer_input_capture_config	configure TIMER input capture parameter
timer_channel_input_capture_prescaler_config	configure TIMER channel input capture prescaler value
timer_channel_capture_value_register_read	read TIMER channel capture compare register value
timer_input_pwm_capture_config	configure TIMER input pwm capture function
timer_hall_mode_config	configure TIMER hall sensor mode
timer_input_trigger_source_select	select TIMER input trigger source
timer_master_output_trigger_source_select	select TIMER master mode output trigger source
timer_slave_mode_select	select TIMER slave mode
timer_master_slave_mode_config	configure TIMER master slave mode
timer_external_trigger_config	configure TIMER external trigger input
timer_quadrature_decoder_mode_config	configure TIMER quadrature decoder mode
timer_internal_clock_config	configure TIMER internal clock mode
timer_internal_trigger_as_external_clock_config	configure TIMER the internal trigger as external clock input
timer_external_trigger_as_external_clock_config	configure TIMER the external trigger as external clock input
timer_external_clock_mode0_config	configure TIMER the external clock mode 0
timer_external_clock_mode1_config	configure TIMER the external clock mode 1
timer_external_clock_mode1_disable	disable TIMER the external clock mode 1
timer_write_chxval_register_config	configure TIMER write CHxVAL register selection
timer_output_value_selection_config	configure TIMER output value selection
timer_flag_get	get TIMER flags
timer_flag_clear	clear TIMER flags
timer_interrupt_enable	enable the TIMER interrupt
timer_interrupt_disable	disable the TIMER interrupt
timer_interrupt_flag_get	get timer interrupt flag
timer_interrupt_flag_clear	clear TIMER interrupt flag

Structure timer_parameter_struct

Table 3-893. 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
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 3-894. 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)
breakstate	break enable (TIMER_BREAK_ENABLE, TIMER_BREAK_DISABLE)

Structure timer_oc_parameter_struct

Table 3-895. 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)

Member name	Function description
ocnidlestate	idle state of channel complementary output (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH)

Structure timer_ic_parameter_struct

Table 3-896. 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 3-897. 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 3-898. Function timer_struct_para_init

Function name	timer_struct_para_init
Function prototype	void timer_struct_para_init(timer_parameter_struct* initpara);

Function descriptions		initialize the parameters of TIMER init parameter struct with the default values
Precondition		-
The called functions		-
Input parameter{in}		
initpara	TIMER init parameter struct, the structure members can refer to Structure timer_parameter_struct .	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

```
/* initialize TIMER init parameter struct with a default value */

timer_parameter_struct timer_initpara;

timer_struct_para_init(&timer_initpara);
```

timer_init

The description of timer_init is shown as below:

Table 3-899. 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 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 3-900. Function timer_enable

Function name	timer_enable
Function prototype	void timer_enable(uint32_t timer_periph);
Function descriptions	enable a timer
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable TIMER0 */

timer_enable(TIMER0);
  
```

timer_disable

The description of timer_disable is shown as below:

Table 3-901. Function timer_disable

Function name	timer_disable
Function prototype	void timer_disable(uint32_t timer_periph);
Function descriptions	disable a timer
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 */

timer_disable(TIMER0);
```

timer_auto_reload_shadow_enable

The description of timer_auto_reload_shadow_enable is shown as below:

Table 3-902. 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
TIMERx(x=0..13)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TIMER0 auto reload shadow function */

timer_auto_reload_shadow_enable(TIMER0);
```

timer_auto_reload_shadow_disable

The description of timer_auto_reload_shadow_disable is shown as below:

Table 3-903. Function timer_auto_reload_shadow_disable

Function name	timer_auto_reload_shadow_disable
Function prototype	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
Function descriptions	disable the auto reload shadow function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..13)	TIMER peripheral selection

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 auto reload shadow function */

timer_auto_reload_shadow_disable(TIMER0);
```

timer_update_event_enable

The description of timer_update_event_enable is shown as below:

Table 3-904. Function timer_update_event_enable

Function name	timer_update_event_enable
Function prototype	void timer_update_event_enable(uint32_t timer_periph);
Function descriptions	enable the update event
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER0 the update event */

timer_update_event_enable (TIMER0);
```

timer_update_event_disable

The description of timer_update_event_disable is shown as below:

Table 3-905. 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 3-906. Function timer_counter_alignment

Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..4,7)	TIMER peripheral selection
Input parameter{in}	
aligned	alignment mode
TIMER_COUNTER_EDGE	No center-aligned mode (edge-aligned mode). The direction of the counter is specified by the DIR bit.
TIMER_COUNTER_CENTER_DOWN	Center-aligned and counting down assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0register). Only when the counter is counting down, compare interrupt flag of channels can be set.
TIMER_COUNTER_CENTER_UP	Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0register). Only when the counter is counting up, compare interrupt flag of channels can be set.
TIMER_COUNTER_CENTER_BOTH	Center-aligned and counting up/down assert mode. The counter counts under center-aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0 register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set TIMER0 counter center-aligned and counting up assert mode */
```

```
timer_counter_alignment(TIMER0, TIMER_COUNTER_CENTER_UP);
```

timer_counter_up_direction

The description of timer_counter_up_direction is shown as below:

Table 3-907. Function timer_counter_up_direction

Function name	timer_counter_up_direction
Function prototype	void timer_counter_up_direction(uint32_t timer_periph);
Function descriptions	set TIMER counter up direction
Precondition	set TIMER counter no center-aligned mode (edge-aligned mode)
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..4,7)	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 3-908. timer_counter_down_direction

Function name	timer_counter_down_direction
Function prototype	void timer_counter_down_direction(uint32_t timer_periph);
Function descriptions	set TIMER counter down direction
Precondition	set TIMER counter no center-aligned mode (edge-aligned mode)
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..4,7)	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 3-909. Function timer_prescaler_config

Function name	timer_prescaler_config
Function prototype	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint32_t pscreload);
Function descriptions	configure TIMER prescaler
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Input parameter{in}	
prescaler	prescaler value (0~65535)
Input parameter{in}	
pscreload	prescaler reload mode
<i>TIMER_PSC_RELOAD_NOW</i>	the prescaler is loaded right now
<i>TIMER_PSC_RELOAD_UPDATE</i>	the prescaler is loaded at the next update event
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 prescaler */

timer_prescaler_config(TIMER0, 3000, TIMER_PSC_RELOAD_NOW);

```

timer_repetition_value_config

The description of timer_repetition_value_config is shown as below:

Table 3-910. Function timer_repetition_value_config

Function name	timer_repetition_value_config
Function prototype	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
Function descriptions	configure TIMER repetition register value
Precondition	-

The called functions		-
Input parameter{in}		
timer_periph		TIMER peripheral
<i>TIMERx(x=0,7)</i>		TIMER peripheral selection
Input parameter{in}		
repetition		the counter repetition value (0~255)
Output parameter{out}		
-		-
Return value		
-		-

Example:

```
/* configure TIMER0 repetition register value */

timer_repetition_value_config(TIMER0, 98);
```

timer_autoreload_value_config

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

Table 3-911. 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 (0-0xFFFFFFFF)	
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 3-912. 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 (0-0xFFFFFFFF)
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 3-913. 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(0~0xFFFFFFFF)

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 3-914. Function timer_prescaler_read

Function name	timer_prescaler_read
Function prototype	uint16_t timer_prescaler_read(uint32_t timer_periph);
Function descriptions	read TIMER prescaler value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
uint16_t	prescaler register value (0x0000~0xFFFF)

Example:

```
/* read TIMER0 prescaler value */

uint16_t i = 0;

i = timer_prescaler_read(TIMER0);
```

timer_single_pulse_mode_config

The description of timer_single_pulse_mode_config is shown as below:

Table 3-915. Function timer_single_pulse_mode_config

Function name	timer_single_pulse_mode_config
Function prototype	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
Function descriptions	configure TIMER single pulse mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..8,11)</i>	TIMER peripheral selection
Input parameter{in}	
spmode	pulse mode
<i>TIMER_SP_MODE_SIN GLE</i>	single pulse mode
<i>TIMER_SP_MODE_RE PETITIVE</i>	repetitive pulse mode

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 single pulse mode */
timer_single_pulse_mode_config(TIMER0, TIMER_SP_MODE_SINGLE);
```

timer_update_source_config

The description of timer_update_source_config is shown as below:

Table 3-916. Function timer_update_source_config

Function name	timer_update_source_config
Function prototype	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
Function descriptions	configure TIMER update source
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Input parameter{in}	
update	update source
<i>TIMER_UPDATE_SRC_GLOBAL</i>	Any of the following events generate an update interrupt or DMA request: <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_REGULAR</i>	Only counter overflow/underflow generates an update interrupt or DMA request.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER update only by counter overflow/underflow */
timer_update_source_config(TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

timer_dma_enable

The description of timer_dma_enable is shown as below:

Table 3-917. Function timer_dma_enable

Function name	timer_dma_enable
Function prototype	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
Function descriptions	enable the TIMER DMA
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
dma	timer DMA source enable
<i>TIMER_DMA_UPD</i>	update DMA enable, TIMERx(x=0..7)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable, TIMERx(x=0,7)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable, TIMERx(x=0..4,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TIMER0 update DMA */
timer_dma_enable(TIMER0, TIMER_DMA_UPD);
```

timer_dma_disable

The description of timer_dma_disable is shown as below:

Table 3-918. Function timer_dma_disable

Function name	timer_dma_disable
Function prototype	void timer_dma_disable (uint32_t timer_periph, uint16_t dma);
Function descriptions	disable the TIMER DMA
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters

Input parameter{in}	
dma	timer DMA source disable
<i>TIMER_DMA_UPD</i>	update DMA disable, TIMERx(x=0..7)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA disable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA disable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA disable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA disable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CMTD</i>	commutation DMA request disable, TIMERx(x=0..7)
<i>TIMER_DMA_TRGD</i>	trigger DMA disable, TIMERx(x=0..4,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 update DMA */
timer_dma_disable(TIMER0, TIMER_DMA_UPD);
```

timer_channel_dma_request_source_select

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

Table 3-919. Function timer_channel_dma_request_source_select

Function name	timer_channel_dma_request_source_select
Function prototype	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
Function descriptions	channel DMA request source selection
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection
Input parameter{in}	
dma_request	channel DMA request source selection
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	DMA request of channel n is sent when channel y event occurs
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	DMA request of channel n is sent when update event occurs
Output parameter{out}	
-	-
Return value	
-	-

Example:

/* TIMER0 channel DMA request of channel n is sent when channel y event occurs */

```
timer_channel_dma_request_source_select(TIMER0,
  TIMER_DMAREQUEST_CHANNELEVENT);
```

timer_dma_transfer_config

The description of timer_dma_transfer_config is shown as below:

Table 3-920. 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}	
<i>timer_periph</i>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<i>dma_baseaddr</i>	DMA transfer access start address
<i>TIMER_DMACFG_DMA_TA_CTL0</i>	DMA transfer address is TIMER_CTL0, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CTL1</i>	DMA transfer address is TIMER_CTL1, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_SMCFG</i>	DMA transfer address is TIMER_SMCFG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_DMAINTEN</i>	DMA transfer address is TIMER_DMAINTEN, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_INTF</i>	DMA transfer address is TIMER_INTF, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_SWEVG</i>	DMA transfer address is TIMER_SWEVG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CHCTL0</i>	DMA transfer address is TIMER_CHCTL0, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CHCTL1</i>	DMA transfer address is TIMER_CHCTL1, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CHCTL2</i>	DMA transfer address is TIMER_CHCTL2, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CNT</i>	DMA transfer address is TIMER_CNT, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_PSC</i>	DMA transfer address is TIMER_PSC, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA_TA_CAR</i>	MA transfer address is TIMER_CAR, TIMERx(x=0..4,7)

TA_CAR	
<i>TIMER_DMACFG_DMA</i> TA_CREP	DMA transfer address is TIMER_CREP, TIMERx(x=0..7)
<i>TIMER_DMACFG_DMA</i> TA_CH0CV	DMA transfer address is TIMER_CH0CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA</i> TA_CH1CV	DMA transfer address is TIMER_CH1CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA</i> TA_CH2CV	DMA transfer address is TIMER_CH2CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA</i> TA_CH3CV	DMA transfer address is TIMER_CH3CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA</i> TA_CCHP	DMA transfer address is TIMER_CCHP, TIMERx(x=0..7)
<i>TIMER_DMACFG_DMA</i> TA_DMACFG	DMA transfer address is TIMER_DMACFG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA</i> TA_DMATB	DMA transfer address is TIMER_DMATB, TIMERx(x=0..4,7)
Input parameter{in}	
dma_lenth	DMA transfer count
<i>TIMER_DMACFG_DMA</i> TC_xTRANSFER	x=1..18, DMA transfer x time
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the TIMER0 DMA transfer */

timer_dma_transfer_config(TIMER0, TIMER_DMACFG_DMATA_CTL0,
  TIMER_DMACFG_DMATC_5TRANSFER);
```

timer_event_software_generate

The description of timer_event_software_generate is shown as below:

Table 3-921. 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
TIMERx	please refer to the following parameters
Input parameter{in}	
event	the timer software event generation sources
TIMER_EVENT_SRC_UPG	update event,TIMERx(x=0..13)
TIMER_EVENT_SRC_C_H0G	channel 0 capture or compare event generation,TIMERx(x=0..4,7..13)
TIMER_EVENT_SRC_C_H1G	channel 1 capture or compare event generation,TIMERx(x=0..4,7,8,11)
TIMER_EVENT_SRC_C_H2G	channel 2 capture or compare event generation,TIMERx(x=0..4,7)
TIMER_EVENT_SRC_C_H3G	channel 3 capture or compare event generation,TIMERx(x=0..4,7)
TIMER_EVENT_SRC_C_MTG	channel commutation event generation,TIMERx(x=0,7)
TIMER_EVENT_SRC_T_RGG	trigger event generation,TIMERx(x=0..4,7,8,11)
TIMER_EVENT_SRC_B_RKG	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 3-922. Function `timer_break_struct_para_init`

Function name	timer_break_struct_para_init
Function prototype	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
Function descriptions	initialize the parameters of TIMER break parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
breakpara	TIMER break parameter struct, the structure members can refer to

	<u>Structure timer_break_parameter_struct.</u>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER break parameter struct with a default value */
```

```
timer_break_parameter_struct timer_breakpara;
```

```
timer_break_struct_para_init(&timer_breakpara);
```

timer_break_config

The description of timer_break_config is shown as below:

Table 3-923. 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 <u>Structure timer_break_parameter_struct.</u>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 break function */
```

```
timer_break_parameter_struct timer_breakpara;
```

```
timer_breakpara.runoffstate = TIMER_ROS_STATE_DISABLE;
```

```
timer_breakpara.ideloffstate = TIMER_IOS_STATE_DISABLE ;
```

```
timer_breakpara.deadtime = 255;
```

```
timer_breakpara.breakpolarity = TIMER_BREAK_POLARITY_LOW;
```

```

timer_breakpara.outputautostate = TIMER_OUTAUTO_ENABLE;
timer_breakpara.protectmode = TIMER_CCHP_PROT_0;
timer_breakpara.breakstate = TIMER_BREAK_ENABLE;
timer_break_config(TIMER0, &timer_breakpara);
  
```

timer_break_enable

The description of timer_break_enable is shown as below:

Table 3-924. 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 3-925. 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` is shown as below:

Table 3-926. 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-filled in TIMERx_CCHP register is 00.
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER0 output automatic function */
timer_automatic_output_enable(TIMER0);
```

timer_automatic_output_disable

The description of `timer_automatic_output_disable` is shown as below:

Table 3-927. 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-filled 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 3-928. Function timer_primary_output_config

Function name	timer_primary_output_config
Function prototype	void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue);
Function descriptions	configure TIMER primary output function
Precondition	-
The called functions	-
Input parameter{in}	
<i>timer_periph</i>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
<i>newvalue</i>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable 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 3-929. Function timer_channel_control_shadow_config

Function name	timer_channel_control_shadow_config
Function prototype	void timer_channel_control_shadow_config(uint32_t timer_periph,

	ControlStatus newvalue);
Function descriptions	channel commutation control shadow register enable
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
newvalue	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* channel capture/compare control shadow register enable */
timer_channel_control_shadow_config(TIMER0, ENABLE);
```

timer_channel_control_shadow_update_config

The description of `timer_channel_control_shadow_update_config` is shown as below:

Table 3-930. 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 commutation control shadow register update control
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
ccuctl	channel control shadow register update control
<i>TIMER_UPDATECTL_CU</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 3-931. 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* ocpars);
Function descriptions	initialize the parameters of TIMER channel output parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
ocpara	TIMER channel output parameter struct, the structure members can refer to Structure timer_oc_parameter_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER channel output parameter struct with a default value */
timer_oc_parameter_struct timer_ocinitpara;
timer_channel_output_struct_para_init(&timer_ocinitpara);
```

timer_channel_output_config

The description of timer_channel_output_config is shown as below:

Table 3-932. 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* ocpars);
Function descriptions	configure TIMER channel output function
Precondition	-
The called functions	-
Input parameter{in}	

timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
ocpara	TIMER channel output parameter struct, the structure members can refer to <u>Structure timer_oc_parameter_struct</u> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output function */

timer_oc_parameter_struct timer_ocinitpara;

timer_ocinitpara.outputstate = TIMER_CCX_ENABLE;

timer_ocinitpara.outputnstate = TIMER_CCXN_ENABLE;

timer_ocinitpara.ocpolarity = TIMER_OC_POLARITY_HIGH;

timer_ocinitpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;

timer_ocinitpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;

timer_ocinitpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;

timer_channel_output_config(TIMER0, TIMER_CH_0, &timer_ocinitpara);
```

timer_channel_output_mode_config

The description of `timer_channel_output_mode_config` is shown as below:

Table 3-933. Function `timer_channel_output_mode_config`

Function name	<code>timer_channel_output_mode_config</code>
Function prototype	<code>void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);</code>
Function descriptions	configure TIMER channel output compare mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral

<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
ocmode	channel output compare mode
<i>TIMER_OC_MODE_TIMING</i>	timing mode
<i>TIMER_OC_MODE_ACITIVE</i>	set the channel output
<i>TIMER_OC_MODE_INACTIVE</i>	clear the channel output
<i>TIMER_OC_MODE_TOGGLEGGLE</i>	toggle on match
<i>TIMER_OC_MODE_FORCELOW</i>	force low mode
<i>TIMER_OC_MODE_FORCEHIGH</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 3-934. Function `timer_channel_output_pulse_value_config`

Function name	<code>timer_channel_output_pulse_value_config</code>
Function prototype	<code>void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);</code>
Function descriptions	configure TIMER channel output pulse value

Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMERx(x=0..4,7..13))
TIMER_CH_1	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
TIMER_CH_2	TIMER channel 2 (TIMERx(x=0..4,7))
TIMER_CH_3	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
pulse	channel output pulse value (0~0xFFFFFFFF)
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 3-935. 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
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMERx(x=0..4,7..13))
TIMER_CH_1	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
TIMER_CH_2	TIMER channel 2 (TIMERx(x=0..4,7))
TIMER_CH_3	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	

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 3-936. Function `timer_channel_output_fast_config`

Function name	<code>timer_channel_output_fast_config</code>
Function prototype	<code>void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);</code>
Function descriptions	configure TIMER channel output fast function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (<i>TIMERx</i> (x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (<i>TIMERx</i> (x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (<i>TIMERx</i> (x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (<i>TIMERx</i> (x=0..4,7))
Input parameter{in}	
ocfast	channel output fast function
<i>TIMER_OC_FAST_ENA</i> <i>BLE</i>	channel output fast function enable
<i>TIMER_OC_FAST_DIS</i> <i>ABLE</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 3-937. Function timer_channel_output_clear_config

Function name	timer_channel_output_clear_config
Function prototype	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
Function descriptions	configure TIMER channel output clear function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER periphera
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMERx(x=0..4,7..13))
TIMER_CH_1	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
TIMER_CH_2	TIMER channel 2 (TIMERx(x=0..4,7))
TIMER_CH_3	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
occlear	channel output clear function
TIMER_OC_CLEAR_ENABLE	channel output clear function enable
TIMER_OC_CLEAR_DISABLE	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,
TICKER_OC_CLEAR_ENABLE);
```

timer_channel_output_polarity_config

The description of timer_channel_output_polarity_config is shown as below:

Table 3-938. 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>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
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,
TICKER_OC_POLARITY_HIGH);
```

timer_channel_complementary_output_polarity_config

The description of timer_channel_complementary_output_polarity_config is shown as below:

Table 3-939. 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
TIMERx(x=0,7)	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0
TIMER_CH_1	TIMER channel 1
TIMER_CH_2	TIMER channel 2
Input parameter{in}	
ocpolarity	channel complementary output polarity
TIMER_OCN_POLARITY_H	channel complementary output polarity is high
TIMER_OCN_POLARITY_L	channel complementary output polarity is low
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output polarity */

timer_channel_complementary_output_polarity_config(TIMER0, TIMER_CH_0,
TICKER_OCN_POLARITY_HIGH);
```

timer_channel_output_state_config

The description of `timer_channel_output_state_config` is shown as below:

Table 3-940. 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
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured

<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
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 3-941. Function `timer_channel_complementary_output_state_config`

Function name	timer_channel_complementary_output_state_config
Function prototype	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
Function descriptions	configure TIMER channel complementary output enable state
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<i>TIMER_CH_2</i>	TIMER channel 2
Input parameter{in}	
state	TIMER channel complementary output enable state
<i>TIMER_CCXN_ENABLE</i>	channel complementary enable
<i>TIMER_CCXN_DISABLE</i>	channel complementary disable
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output enable state */

timer_channel_complementary_output_state_config(TIMER0, TIMER_CH_0,
TIMER_CCXN_ENABLE);
```

timer_channel_input_struct_para_init

The description of timer_channel_input_struct_para_init is shown as below:

Table 3-942. Function timer_channel_input_struct_para_init

Function name	timer_channel_input_struct_para_init
Function prototype	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
Function descriptions	initialize the parameters of TIMER channel input parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
icpara	TIMER channel input parameter struct, the structure members can refer to Structure timer_ic_parameter_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER channel input parameter struct with a default value */

timer_ic_parameter_struct timer_icinitpara;

timer_channel_input_struct_para_init(&timer_icinitpara);
```

timer_input_capture_config

The description of timer_input_capture_config is shown as below:

Table 3-943. 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>		please refer to the following parameters
Input parameter{in}		
channel		channel to be configured
<i>TIMER_CH_0</i>		TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>		TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>		TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>		TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}		
icpara		TIMER channel input parameter struct, the structure members can refer to <u>Structure timer_ic_parameter_struct</u>
Output parameter{out}		
-		-
Return value		
-		-

Example:

```
/* configure TIMER0 input capture parameter */
timer_ic_parameter_struct timer_icinitpara;
timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;
timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
```

timer_channel_input_capture_prescaler_config

The description of timer_channel_input_capture_prescaler_config is shown as below:

Table 3-944. Function timer_channel_input_capture_prescaler_config

Function name	timer_channel_input_capture_prescaler_config
Function prototype	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);
Function descriptions	configure TIMER channel input capture prescaler value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters

Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
prescaler	channel input capture prescaler value
<i>TIMER_IC_PSC_DIV1</i>	no prescaler
<i>TIMER_IC_PSC_DIV2</i>	divided by 2
<i>TIMER_IC_PSC_DIV4</i>	divided by 4
<i>TIMER_IC_PSC_DIV8</i>	divided by 8
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 input capture prescaler value */
timer_channel_input_capture_prescaler_config(TIMER0, TIMER_CH_0,
TICK_IC_PSC_DIV2);
```

timer_channel_capture_value_register_read

The description of `timer_channel_capture_value_register_read` is shown as below:

Table 3-945. Function `timer_channel_capture_value_register_read`

Function name	<code>timer_channel_capture_value_register_read</code>
Function prototype	<code>uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);</code>
Function descriptions	read TIMER channel capture compare register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Output parameter{out}	

-	-
Return value	
uint32_t	channel capture compare register value (0~0xFFFFFFFF)

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 3-946. 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
TIMERx(x=0..4,7,8,11)	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0
TIMER_CH_1	TIMER channel 1
Input parameter{in}	
icpwm	TIMER channel input pwm parameter struct, the structure members can refer to Structure timer_ic_parameter_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 input pwm capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
```

```

timer_icinitpara.icfilter      = 0x0;
timer_input_pwm_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
  
```

timer_hall_mode_config

The description of timer_hall_mode_config is shown as below:

Table 3-947. Function timer_hall_mode_config

Function name	timer_hall_mode_config
Function prototype	void timer_hall_mode_config(uint32_t timer_periph, uint32_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_CE_ENABLE	TIMER hall sensor mode enable
TIMER_HALLINTERFACE_CE_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 3-948. 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>	please refer to the following parameters
Input parameter{in}	
intrigger	trigger selection
<i>TIMER_SMCFG_TRGS_EL_ITI0</i>	Internal trigger input 0 (ITI0, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS_EL_ITI1</i>	Internal trigger input 0 (ITI1, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS_EL_ITI2</i>	Internal trigger input 0 (ITI2, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS_EL_ITI3</i>	Internal trigger input 0 (ITI3, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS_EL_CI0F_ED</i>	CI0 edge flag (CI0F_ED, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS_EL_CI0FE0</i>	channel 0 input Filtered output (CI0FE0, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS_EL_CI1FE1</i>	channel 1 input Filtered output (CI1FE1, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS_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 3-949. Function timer_master_output_trigger_source_select

Function name	timer_master_output_trigger_source_select
Function prototype	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
Function descriptions	select TIMER master mode output trigger source
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	

outtrigger	master mode control
<i>TIMER_TRI_OUT_SRC_RESET</i>	Reset. When the UPG bit in the TIMERx_SWEVG register is set or a reset is generated by the slave mode controller, a TRGO pulse occurs. And in the latter case, the signal on TRGO is delayed compared to the actual reset
<i>TIMER_TRI_OUT_SRC_ENABLE</i>	Enable. This mode is useful to start several timers at the same time or to control a window in which a slave timer is enabled. In this mode the master mode controller selects the counter enable signal as TRGO. The counter enable signal is set when CEN control bit is set or the trigger input in pause mode is high. There is a delay between the trigger input in pause mode and the TRGO output, except if the master-slave mode is selected.
<i>TIMER_TRI_OUT_SRC_UPDATE</i>	Update. In this mode the master mode controller selects the update event as TRGO.
<i>TIMER_TRI_OUT_SRC_CC0</i>	Capture/compare pulse. In this mode the master mode controller generates a TRGO pulse when a capture or a compare match occurred in channel 0.
<i>TIMER_TRI_OUT_SRC_O0CPRE</i>	Compare. In this mode the master mode controller selects the O0CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O1CPRE</i>	Compare. In this mode the master mode controller selects the O1CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O2CPRE</i>	Compare. In this mode the master mode controller selects the O2CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O3CPRE</i>	Compare. In this mode the master mode controller selects the O3CPRE signal is used as TRGO.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select TIMER0 master mode output trigger source */
timer_master_output_trigger_source_select (TIMER0, TIMER_TRI_OUT_SRC_RESET);
```

timer_slave_mode_select

The description of timer_slave_mode_select is shown as below:

Table 3-950. Function timer_slave_mode_select

Function name	timer_slave_mode_select
Function prototype	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
Function descriptions	select TIMER slave mode
Precondition	-
The called functions	-
Input parameter{in}	

timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
Input parameter{in}	
slavemode	slave mode
<i>TIMER_SLAVE_MODE_DISABLE</i>	slave mode disable
<i>TIMER_QUAD_DECODER_MODE0</i>	quadrature decoder mode 0
<i>TIMER_QUAD_DECODER_MODE1</i>	quadrature decoder mode 1
<i>TIMER_QUAD_DECODER_MODE2</i>	quadrature decoder mode 2
<i>TIMER_SLAVE_MODE_RESTART</i>	restart mode
<i>TIMER_SLAVE_MODE_PAUSE</i>	pause mode
<i>TIMER_SLAVE_MODE_EVENT</i>	event mode
<i>TIMER_SLAVE_MODE_EXTERNAL0</i>	external clock mode 0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select TIMER0 slave mode */

timer_slave_mode_select (TIMER0, TIMER_QUAD_DECODER_MODE0);
```

timer_master_slave_mode_config

The description of `timer_master_slave_mode_config` is shown as below:

Table 3-951. Function `timer_master_slave_mode_config`

Function name	timer_master_slave_mode_config
Function prototype	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);
Function descriptions	configure TIMER master slave mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<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 3-952. Function timer_external_trigger_config

Function name	timer_external_trigger_config
Function prototype	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
Function descriptions	configure TIMER external trigger input
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
Input parameter{in}	
extprescaler	external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
Input parameter{in}	
expolarity	external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active

Input parameter{in}	
extfilter	external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 external trigger input */

timer_external_trigger_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,
  TIMER_ETP_FALLING, 10);
```

timer_quadrature_decoder_mode_config

The description of timer_quadrature_decoder_mode_config is shown as below:

Table 3-953. Function timer_quadrature_decoder_mode_config

Function name	timer_quadrature_decoder_mode_config
Function prototype	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
Function descriptions	configure TIMER quadrature decoder mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
Input parameter{in}	
decomode	quadrature decoder mode
<i>TIMER_QUAD_DECODER_MODE0</i>	counter counts on CI0FE0 edge depending on CI1FE1 level
<i>TIMER_QUAD_DECODER_MODE1</i>	counter counts on CI1FE1 edge depending on CI0FE0 level
<i>TIMER_QUAD_DECODER_MODE2</i>	counter counts on both CI0FE0 and CI1FE1 edges depending on the level of the other input
Input parameter{in}	
ic0polarity	IC0 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	active both 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
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	active both 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 3-954. 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 3-955. Function `timer_internal_trigger_as_external_clock_config`

Function name	timer_internal_trigger_as_external_clock_config
Function prototype	<code>void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);</code>
Function descriptions	configure TIMER the internal trigger as external clock input
Precondition	-
The called functions	<code>timer_input_trigger_source_select</code>
Input parameter{in}	
timer_periph	TIMER peripheral
<code>TIMERx(x=0..4,7,8,11)</code>	TIMER peripheral selection
Input parameter{in}	
intrigger	trigger selection
<code>TIMER_SMCFG_TRGS_EL_ITI0</code>	Internal trigger input 0 (ITI0)
<code>TIMER_SMCFG_TRGS_EL_ITI1</code>	Internal trigger input 0 (ITI1)
<code>TIMER_SMCFG_TRGS_EL_ITI2</code>	Internal trigger input 0 (ITI2)
<code>TIMER_SMCFG_TRGS_EL_ITI3</code>	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 3-956. Function `timer_external_trigger_as_external_clock_config`

Function name	timer_external_trigger_as_external_clock_config
Function prototype	<code>void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter);</code>
Function descriptions	configure TIMER the external trigger as external clock input
Precondition	-
The called functions	<code>timer_input_trigger_source_select</code>
Input parameter{in}	
timer_periph	TIMER peripheral
<code>TIMERx(x=0..4,7,8,11)</code>	TIMER peripheral selection

Input parameter{in}	
extrigger	external trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI0F_ED</i>	CI0 edge flag (CI0F_ED)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI0FE0</i>	channel 0 input Filtered output (CI0FE0)
<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_RISING</i>	active high or rising edge active
<i>TIMER_IC_POLARITY_FALLING</i>	active low or falling edge active
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	active both edge
Input parameter{in}	
extfilter	external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 the external trigger CI0FE0 as external clock input */

timer_external_trigger_as_external_clock_config(TIMER0,
  TIMER_SMCFG_TRGSEL_CI0FE0, TIMER_IC_POLARITY_RISING, 0);
```

timer_external_clock_mode0_config

The description of timer_external_clock_mode0_config is shown as below:

Table 3-957. Function timer_external_clock_mode0_config

Function name	timer_external_clock_mode0_config
Function prototype	void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
Function descriptions	configure TIMER the external clock mode0
Precondition	-
The called functions	timer_external_trigger_config
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
Input parameter{in}	

extprescaler	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
Input parameter{in}	
expolarity	ETI external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
Input parameter{in}	
extfilter	ETI external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 the external clock mode0 */
timer_external_clock_mode0_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 0);
```

timer_external_clock_mode1_config

The description of timer_external_clock_mode1_config is shown as below:

Table 3-958. Function timer_external_clock_mode1_config

Function name	timer_external_clock_mode1_config
Function prototype	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
Function descriptions	configure TIMER the external clock mode1
Precondition	-
The called functions	timer_external_trigger_config
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
Input parameter{in}	
extprescaler	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided

<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
Input parameter{in}	
expolarity	ETI external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
Input parameter{in}	
extfilter	ETI external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 the external clock mode1 */
timer_external_clock_mode1_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 0);
```

timer_external_clock_mode1_disable

The description of `timer_external_clock_mode1_disable` is shown as below:

Table 3-959. Function `timer_external_clock_mode1_disable`

Function name	timer_external_clock_mode1_disable
Function prototype	void timer_external_clock_mode1_disable(uint32_t timer_periph);
Function descriptions	disable TIMER the external clock mode1
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 the external clock mode1 */
timer_external_clock_mode1_disable(TIMER0);
```

```
timer_external_clock_mode1_disable(TIMER0);
```

timer_write_chxval_register_config

The description of timer_write_chxval_register_config is shown as below:

Table 3-960. Function timer_write_chxval_register_config

Function name	timer_write_chxval_register_config
Function prototype	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
Function descriptions	configure TIMER write CHxVAL register selection
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7..13)</i>	TIMER peripheral selection
Input parameter{in}	
ccsel	write CHxVAL register selection
<i>TIMER_CHVSEL_DISA BLE</i>	no effect
<i>TIMER_CHVSEL_ENAB LE</i>	when write the CHxVAL register, if the write value is same as the CHxVAL value, the write access is ignored
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 write CHxVAL register selection */
timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

timer_output_value_selection_config

The description of timer_output_value_selection_config is shown as below:

Table 3-961. Function timer_output_value_selection_config

Function name	timer_output_value_selection_config
Function prototype	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);
Function descriptions	configure TIMER output value selection
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx (x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	

outsel	output value selection
<i>TIMER_OUTSEL_DISA BLE</i>	no effect
<i>TIMER_OUTSEL_ENAB LE</i>	if POEN and IOS is 0, the output disabled
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER output value selection */
timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

timer_flag_get

The description of timer_flag_get is shown as below:

Table 3-962. Function timer_flag_get

Function name	timer_flag_get
Function prototype	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
Function descriptions	get TIMER flags
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
flag	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag, TIMERx(x=0..13)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, TIMERx(x=0..4,7..13)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, TIMERx(x=0..4,7,8,11)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, TIMERx(x=0..4,7)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, TIMERx(x=0..4,7)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, TIMERx(x=0,7)
<i>TIMER_FLAG_TRG</i>	trigger flag, TIMERx(x=0,7,8,11)
<i>TIMER_FLAG_BRK</i>	break flag, TIMERx(x=0,7)
<i>TIMER_FLAG_CH00</i>	channel 0 overcapture flag, TIMERx(x=0..4,7..11)
<i>TIMER_FLAG_CH10</i>	channel 1 overcapture flag, TIMERx(x=0..4,7,8,11)
<i>TIMER_FLAG_CH20</i>	channel 2 overcapture flag, TIMERx(x=0..4,7)
<i>TIMER_FLAG_CH30</i>	channel 3 overcapture flag, TIMERx(x=0..4,7)
Output parameter{out}	
-	-

Return value	
FlagStatus	SET or RESET

Example:

```
/* get TIMER0 update flags */

FlagStatus Flag_status = RESET;

Flag_status = timer_flag_get(TIMER0, TIMER_FLAG_UP);
```

timer_flag_clear

The description of timer_flag_clear is shown as below:

Table 3-963. 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
TIMERx	please refer to the following parameters
Input parameter{in}	
flag	the timer interrupt flags
TIMER_FLAG_UP	update flag,TIMERx(x=0..13)
TIMER_FLAG_CH0	channel 0 flag,TIMERx(x=0..4,7..13)
TIMER_FLAG_CH1	channel 1 flag,TIMERx(x=0..4,7,8,11)
TIMER_FLAG_CH2	channel 2 flag,TIMERx(x=0..4,7)
TIMER_FLAG_CH3	channel 3 flag,TIMERx(x=0..4,7)
TIMER_FLAG_CMT	channel commutation flag,TIMERx(x=0,7)
TIMER_FLAG_TRG	trigger flag,TIMERx(x=0,7,8,11)
TIMER_FLAG_BRK	break flag,TIMERx(x=0,7)
TIMER_FLAG_CH0O	channel 0 overcapture flag,TIMERx(x=0..4,7..11)
TIMER_FLAG_CH1O	channel 1 overcapture flag,TIMERx(x=0..4,7,8,11)
TIMER_FLAG_CH2O	channel 2 overcapture flag,TIMERx(x=0..4,7)
TIMER_FLAG_CH3O	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 3-964. Function timer_interrupt_enable

Function name	timer_interrupt_enable
Function prototype	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
Function descriptions	enable the TIMER interrupt
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
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 3-965. 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
TIMERx	please refer to the following parameters
Input parameter{in}	
interrupt	timer interrupt disable source
TIMER_INT_UP	update interrupt disable, TIMERx(x=0..13)
TIMER_INT_CH0	channel 0 interrupt disable, TIMERx(x=0..4,7..13)
TIMER_INT_CH1	channel 1 interrupt disable, TIMERx(x=0..4,7,8,11)
TIMER_INT_CH2	channel 2 interrupt disable, TIMERx(x=0..4,7)
TIMER_INT_CH3	channel 3 interrupt disable, TIMERx(x=0..4,7)
TIMER_INT_CMT	commutation interrupt disable, TIMERx(x=0,7)
TIMER_INT_TRG	trigger interrupt disable, TIMERx(x=0..4,7,8,11)
TIMER_INT_BRK	break interrupt disable, TIMERx(x=0,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 update interrupt */

timer_interrupt_disable(TIMER0, TIMER_INT_UP);
```

timer_interrupt_flag_get

The description of timer_interrupt_flag_get is shown as below:

Table 3-966. Function timer_interrupt_flag_get

Function name	timer_interrupt_flag_get
Function prototype	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t int_flag);
Function descriptions	get timer interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
int_flag	the timer interrupt bits
TIMER_INT_FLAG_UP	update interrupt flag,TIMERx(x=0..13)
TIMER_INT_FLAG_CH0	channel 0 interrupt flag,TIMERx(x=0..4,7..13)
TIMER_INT_FLAG_CH1	channel 1 interrupt flag,TIMERx(x=0..4,7,8,11)
TIMER_INT_FLAG_CH2	channel 2 interrupt flag,TIMERx(x=0..4,7)
TIMER_INT_FLAG_CH3	channel 3 interrupt flag,TIMERx(x=0..4,7)

<i>TIMER_INT_FLAG_CM</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	
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 3-967. Function `timer_interrupt_flag_clear`

Function name	timer_interrupt_flag_clear
Function prototype	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t int_flag);
Function descriptions	clear TIMER interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
int_flag	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag,TIMERx(x=0..13)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag,TIMERx(x=0..4,7..13)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag,TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag,TIMERx(x=0..4,7)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag,TIMERx(x=0..4,7)
<i>TIMER_INT_FLAG_CM</i>	channel commutation interrupt flag,TIMERx(x=0,7)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag,TIMERx(x=0,7,8,11)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag,TIMERx(x=0,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TIMER0 update interrupt flag */

timer_interrupt_flag_clear(TIMER0, TIMER_INT_FLAG_UP);
```

3.27. TMU

The Trigonometric Math Unit (TMU) is a fully configurable block that execute common trigonometric and arithmetic operations. The TMU calculation unit can be used to calculate total 9 kinds of operations. The operation data must meet IEEE 32-Bit Single Precision Floating-Point Format. The TMU registers are listed in chapter [3.27.1](#), the TMU firmware functions are introduced in chapter [3.27.2](#).

3.27.1. Descriptions of Peripheral registers

TMU registers are listed in the table shown as below:

Table 3-968. TMU Registers

Registers	Descriptions
TMU_IDATA0	input data0 register
TMU_IDATA1	input data1 register
TMU_CTL	control register
TMU_DATA0	data0 register
TMU_DATA1	data1 register
TMU_STAT	status register

3.27.2. Descriptions of Peripheral functions

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

Table 3-969. TMU firmware function

Function name	Function description
tmu_deinit	reset the TMU
tmu_enable	enable the TMU
tmu_mode_set	configure the TMU mode
tmu_idata0_write	write the data to TMU input data0 regiset
tmu_idata1_write	write the data to TMU input data1 regiset
tmu_data0_read	read the data from TMU data0 regiset
tmu_data1_read	read the data from TMU data1 regiset
tmu_interrupt_enable	enable TTMU interrupt
tmu_interrupt_disable	disable TTMU interrupt
tmu_flag_get	check teh TMU status flag
tmu_interrupt_flag_get	check teh TMU interrupt flag

tmu_deinit

The description of tmu_deinit is shown as below:

Table 3-970. Function tmu_deinit

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

Example:

```
/* reset the TMU */

tmu_deinit();
```

tmu_enable

The description of tmu_enable is shown as below:

Table 3-971. Function tmu_enable

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

Example:

```
/* enable the TMU */

void tmu_enable(void);
```

tmu_mode_set

The description of tmu_mode_set is shown as below:

Table 3-972. Function tmu_mode_set

Function name	tmu_mode_set
Function prototype	void tmu_mode_set(uint32_t mode);
Function descriptions	configure the TMU mode
Precondition	-
The called functions	-
Input parameter{in}	
mode	the operation mode of TMU
<i>TMU_MODE0</i>	the operation mode0
<i>TMU_MODE1</i>	the operation mode1
<i>TMU_MODE2</i>	the operation mode2
<i>TMU_MODE3</i>	the operation mode3
<i>TMU_MODE4</i>	the operation mode4
<i>TMU_MODE5</i>	the operation mode5
<i>TMU_MODE6</i>	the operation mode6
<i>TMU_MODE7</i>	the operation mode7
<i>TMU_MODE8</i>	the operation mode8
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the TMU mode0 */
tmu_mode_set(TMU_MODE0);
```

tmu_idata0_write

The description of tmu_idata0_write is shown as below:

Table 3-973. Function tmu_idata0_write

Function name	tmu_idata0_write
Function prototype	void tmu_idata0_write(uint32_t idata0);
Function descriptions	write the data to TMU input data0 regiset
Precondition	-
The called functions	-
Input parameter{in}	
idata0	the value write to input data0
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* write the idata0 register */

tmu_idata0_write (0XBF000000);
```

tmu_idata1_write

The description of tmu_idata1_write is shown as below:

Table 3-974. Function tmu_idata1_write

Function name	tmu_idata1_write
Function prototype	void tmu_idata1_write(uint32_t idata1);
Function descriptions	write the data to TMU input data1 regiset
Precondition	-
The called functions	-
Input parameter{in}	
idata1	the value write to input data1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write the idata1 register */

tmu_idata1_write (0XBF000000);
```

tmu_data0_read

The description of tmu_data0_read is shown as below:

Table 3-975. Function tmu_data0_read

Function name	tmu_data0_read
Function prototype	uint32_t tmu_data0_read(void);
Function descriptions	read the data from TMU data0 regiset
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the value of data0 register meet IEEE 32-Bit Single Precision Floating-Point

Example:

```

/* read the data from TMU data0 regisetr*/
uint32_t tmu_value = 0;
tmu_value = tmu_data0_read();

```

tmu_data1_read

The description of tmu_data1_read is shown as below:

Table 3-976. Function tmu_data1_read

Function name	tmu_data1_read
Function prototype	uint32_t tmu_data1_read(void);
Function descriptions	read the data from TMU data1 regisetr
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the value of data1 register meet IEEE 32-Bit Single Precision Floating-Point Format

Example:

```

/* read the data from TMU data1 regisetr*/
uint32_t tmu_value = 0;
tmu_value = tmu_data1_read();

```

tmu_interrupt_enable

The description of tmu_interrupt_enable is shown as below:

Table 3-977. Function tmu_interrupt_enable

Function name	tmu_interrupt_enable
Function prototype	void tmu_interrupt_enable(void);
Function descriptions	enable TMU interrupt
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable TMU interrupt */  
  
void tmu_interrupt_enable();
```

tmu_interrupt_disable

The description of `tmu_interrupt_disable` is shown as below:

Table 3-978. Function `tmu_interrupt_disable`

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

Example:

```
/* disable TMU interrupt */  
  
void tmu_interrupt_disable();
```

tmu_flag_get

The description of `tmu_flag_get` is shown as below:

Table 3-979. Function `tmu_flag_get`

Function name	tmu_flag_get
Function prototype	FlagStatus tmu_flag_get(uint32_t flag);
Function descriptions	check teh TMU status flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	teh TMU status flag
TMU_FLAG_OVRF	the flag of TMU overflow
TMU_FLAG_UDRF	the flag of TMU underflow
Output parameter{out}	

-	-
Return value	
FlagStatus	SET/RESET

Example:

```
/* check the TMU overflow flag */

tmu_flag_get(TMU_FLAG_OVRF);
```

tmu_interrupt_flag_get

The description of `tmu_interrupt_flag_get` is shown as below:

Table 3-980. Function `tmu_interrupt_flag_get`

Function name	tmu_interrupt_flag_get
Function prototype	FlagStatus tmu_interrupt_flag_get (uint32_t flag);
Function descriptions	check teh TMU interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	teh TMU interrupt flag
TMU_INT_FLAG_CFIF	the interrupt flag of calculation finished
Output parameter{out}	
-	-
Return value	
FlagStatus	SET/RESET

Example:

```
/* check teh TMU interrupt flag */

tmu_interrupt_flag_getTMU_INT_FLAG_CFIF();
```

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 3-981. USART Registers

Registers	Descriptions
USART_STAT0	Status register 0 (USARTx, x=0..4)

Registers	Descriptions
USART_DATA	Data register (USART _x , x=0..4)
USART_BAUD	Baud rate register (USART _x , x=0..4)
USART_CTL0	Control register 0 (USART _x , x=0..4)
USART_CTL1	Control register 1 (USART _x , x=0..4)
USART_CTL2	Control register 2 (USART _x , x=0..4)
USART_GP	Guard time and prescaler register (USART _x , x=0..4)
USART_CTL3	Control register 3 (USART _x , x=0..4)
USART_RT	Receiver timeout register (USART _x , x=0..4)
USART_STAT1	Status register 1 (USART _x , x=0..4)
USART_GDCTL	GD control register (USART _x , x=0..4)
USART5_CTL0	Control register 0 (USART _x , x=5)
USART5_CTL1	Control register 1 (USART _x , x=5)
USART5_CTL2	Control register 2 (USART _x , x=5)
USART5_BAUD	Baud rate register (USART _x , x=5)
USART5_GP	Guard time and prescaler register (USART _x , x=5)
USART5_RT	Receiver timeout register (USART _x , x=5)
USART5_CMD	Command register (USART _x , x=5)
USART5_STAT	Status register (USART _x , x=5)
USART5_INTC	Interrupt status clear register (USART _x , x=5)
USART5_RDATA	Receive data register (USART _x , x=5)
USART5_TDATA	Transmit data register (USART _x , x=5)
USART5_CHC	Coherence control register (USART _x , x=5)
USART5_RFCS	Receive FIFO control and status register (USART _x , x=5)

3.28.2. Descriptions of Peripheral functions

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

Table 3-982. USART firmware function

Function name	Function description
uart_deinit	reset USART/UART (USART _x , x=0..5)
uart_baudrate_set	configure USART baud rate value (USART _x , x=0..5)
uart parity config	configure USART parity (USART _x , x=0..5)
uart word length set	configure USART word length (USART _x , x=0..5)
uart stop bit set	configure USART stop bit length (USART _x , x=0..5)
uart enable	enable USART (USART _x , x=0..5)
uart disable	disable USART (USART _x , x=0..5)
uart transmit config	configure USART transmitter (USART _x , x=0..5)
uart receive config	configure USART receiver (USART _x , x=0..5)
uart oversample config	configure the USART oversample mode (USART _x , x=0..5)
uart sample bit config	configure sample bit method (USART _x , x=0..5)
uart receiver timeout enable	enable receiver timeout (USART _x , x=0..5)

Function name	Function description
uart_receiver_timeout_disable	disable receiver timeout (USARTx, x=0..5)
uart_receiver_timeout_threshold_config	configure receiver timeout threshold (USARTx, x=0..5)
uart_data_transmit	USART transmit data function (USARTx, x=0..5)
uart_data_receive	USART receive data function (USARTx, x=0..5)
uart_mute_mode_enable	enable mute mode (USARTx, x=0..5)
uart_mute_mode_disable	disable mute mode (USARTx, x=0..5)
uart_mute_mode_wakeup_config	configure wakeup method in mute mode (USARTx, x=0..5)
uart_lin_mode_enable	enable LIN mode (USARTx, x=0..5)
uart_lin_mode_disable	disable LIN mode (USARTx, x=0..5)
uart_lin_break_detection_length_config	configure LIN break frame length (USARTx, x=0..5)
uart_halfduplex_enable	enable half duplex mode (USARTx, x=0..5)
uart_halfduplex_disable	disable half duplex mode (USARTx, x=0..5)
uart_synchronous_clock_enable	enable CK pin in synchronous mode (USARTx, x=0..5)
uart_synchronous_clock_disable	disable CK pin in synchronous mode (USARTx, x=0..5)
uart_synchronous_clock_config	configure USART synchronous mode parameters (USARTx, x=0..5)
uart_guard_time_config	configure guard time value in smartcard mode (USARTx, x=0..5)
uart_smartcard_mode_enable	enable smartcard mode (USARTx, x=0..5)
uart_smartcard_mode_disable	disable smartcard mode (USARTx, x=0..5)
uart_smartcard_mode_nack_enable	enable NACK in smartcard mode (USARTx, x=0..5)
uart_smartcard_mode_nack_disable	disable NACK in smartcard mode (USARTx, x=0..5)
uart_smartcard_autoretry_config	configure smartcard auto-retry number (USARTx, x=0..5)
uart_block_length_config	configure block length (USARTx, x=0..5)
uart_irda_mode_enable	enable IrDA mode (USARTx, x=0..5)
uart_irda_mode_disable	disable IrDA mode (USARTx, x=0..5)
uart_prescaler_config	configure the peripheral clock prescaler in USART IrDA low-power mode (USARTx, x=0..5)
uart_irda_lowpower_config	configure IrDA low-power (USARTx, x=0..5)
uart_dma_receive_config	configure USART DMA reception (USARTx, x=0..5)
uart_dma_transmit_config	configure USART DMA transmission (USARTx, x=0..5)
uart_hardware_flow_rts_config	configure hardware flow control RTS (USARTx, x=0..4)
uart_hardware_flow_cts_config	configure hardware flow control CTS (USARTx, x=0..4)
uart_data_first_config	data is transmitted/received with the LSB/MSB first (USARTx, x=0..4)
uart_invert_config	configure USART inverted (USARTx, x=0..4)
uart_address_config	configure the address of the USART in wake up by address match mode (USARTx, x=0..4)
uart_send_break	send break frame (USARTx, x=0..4)
uart_collision_detected_interrupt_en	enable collision detected interrupt (USARTx, x=0..4)

Function name	Function description
able	
uart_collision_detected_interrupt_disable	disable collision detected interrupt (USARTx, x=0..4)
uart_collision_detection_enable	enable collision detection (USARTx, x=0..4)
uart_collision_detection_disable	disable collision detection (USARTx, x=0..4)
uart_flag_get	get flag in STAT0/STAT1 register (USARTx, x=0..4)
uart_flag_clear	clear flag in STAT0/STAT1 register (USARTx, x=0..4)
uart_interrupt_enable	enable USART interrupt (USARTx, x=0..4)
uart_interrupt_disable	disable USART interrupt (USARTx, x=0..4)
uart_interrupt_flag_get	get USART interrupt flag status (USARTx, x=0..4)
uart_interrupt_flag_clear	clear USART interrupt flag (USARTx, x=0..4)
uart5_data_first_config	data is transmitted/received with the LSB/MSB first (USARTx, x=5)
uart5_invert_config	configure USART5 inverted (USARTx, x=5)
uart5_overrun_enable	enable the USART5 overrun function (USARTx, x=5)
uart5_overrun_disable	disable the USART5 overrun function (USARTx, x=5)
uart5_address_config	configure address of the USART5 (USARTx, x=5)
uart5_address_detection_mode_config	configure address detection mode (USARTx, x=5)
uart5_smartcard_mode_early_nack_enable	enable early NACK in smartcard mode (USARTx, x=5)
uart5_smartcard_mode_early_nack_disable	disable early NACK in smartcard mode (USARTx, x=5)
uart5_reception_error_dma_enable	enable DMA on reception error (USARTx, x=5)
uart5_reception_error_dma_disable	disable DMA on reception error (USARTx, x=5)
uart5_wakeup_enable	enable USART to wakeup the mcu from deep-sleep mode (USARTx, x=5)
uart5_wakeup_disable	disable USART to wakeup the mcu from deep-sleep mode (USARTx, x=5)
uart5_wakeup_mode_config	configure the USART wakeup mode from deep-sleep mode (USARTx, x=5)
uart5_receive_fifo_enable	enable receive FIFO (USARTx, x=5)
uart5_receive_fifo_disable	disable receive FIFO (USARTx, x=5)
uart5_receive_fifo_counter_number	read receive FIFO counter number (USARTx, x=5)
uart5_flag_get	get flag in STAT/RFCs register (USARTx, x=5)
uart5_flag_clear	clear USART status (USARTx, x=5)
uart5_interrupt_enable	enable USART interrupt (USARTx, x=5)
uart5_interrupt_disable	disable USART interrupt (USARTx, x=5)
uart5_command_enable	enable USART command (USARTx, x=5)
uart5_interrupt_flag_get	get USART interrupt and flag status (USARTx, x=5)
uart5_interrupt_flag_clear	clear USART interrupt flag (USARTx, x=5)

Enum `uart_flag_enum`

Table 3-983. `Enum usart_flag_enum`

Member name	Function description
<code>USART_FLAG_CTS</code>	CTS change flag
<code>USART_FLAG_LBD</code>	LIN break detected flag
<code>USART_FLAG_TBE</code>	transmit data buffer empty
<code>USART_FLAG_TC</code>	transmission complete
<code>USART_FLAG_RBNE</code>	read data buffer not empty
<code>USART_FLAG_IDLE</code>	IDLE line detected flag
<code>USART_FLAG_ORERR</code>	overrun error
<code>USART_FLAG_NERR</code>	noise error flag
<code>USART_FLAG_FERR</code>	frame error flag
<code>USART_FLAG_PERR</code>	parity error flag
<code>USART_FLAG_BSY</code>	busy flag
<code>USART_FLAG_EB</code>	end of block flag
<code>USART_FLAG_RT</code>	receiver timeout flag
<code>USART_FLAG_CD</code>	collision detected flag

Enum `uart5_flag_enum`

Table 3-984. `Enum usart5_flag_enum`

Member name	Function description
<code>USART5_FLAG_REA</code>	receive enable acknowledge flag
<code>USART5_FLAG_TEA</code>	transmit enable acknowledge flag
<code>USART5_FLAG_WU</code>	wakeup from deep-sleep mode flag
<code>USART5_FLAG_RWU</code>	receiver wakeup from mute mode
<code>USART5_FLAG_SB</code>	send break flag
<code>USART5_FLAG_AM</code>	ADDR match flag
<code>USART5_FLAG_BSY</code>	busy flag
<code>USART5_FLAG_EB</code>	end of block flag
<code>USART5_FLAG_RT</code>	receiver timeout flag
<code>USART5_FLAG_LBD</code>	LIN break detected flag
<code>USART5_FLAG_TBE</code>	transmit data buffer empty
<code>USART5_FLAG_TC</code>	transmission complete
<code>USART5_FLAG_RBNE</code>	read data buffer not empty
<code>USART5_FLAG_IDLE</code>	IDLE line detected flag
<code>USART5_FLAG_ORERR</code>	overrun error
<code>USART5_FLAG_NERR</code>	noise error flag
<code>USART5_FLAG_FERR</code>	frame error flag
<code>USART5_FLAG_PERR</code>	parity error flag
<code>USART5_FLAG_EPERR</code>	early parity error flag
<code>USART5_FLAG_RFFINT</code>	receive FIFO full interrupt flag

Member name	Function description
USART5_FLAG_RFF	receive FIFO full flag
USART5_FLAG_RFE	receive FIFO empty flag

Enum usart_interrupt_flag_enum

Table 3-985. Enum usart_interrupt_flag_enum

Member name	Function description
USART_INT_FLAG_PERR	parity error interrupt and flag
USART_INT_FLAG_TBE	transmitter buffer empty interrupt and flag
USART_INT_FLAG_TC	transmission complete interrupt and flag
USART_INT_FLAG_RBNE	read data buffer not empty interrupt and 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 and flag
USART_INT_FLAG_LBD	LIN break detected interrupt and flag
USART_INT_FLAG_CTS	CTS interrupt and flag
USART_INT_FLAG_ERR_ORER R	error interrupt and overrun error
USART_INT_FLAG_ERR_NERR	error interrupt and noise error flag
USART_INT_FLAG_ERR_FERR	error interrupt and frame error flag
USART_INT_FLAG_EB	interrupt enable bit of end of block event and flag
USART_INT_FLAG_RT	interrupt enable bit of receive timeout event and flag
USART_INT_FLAG_CD	collision detected interrupt and flag

Enum usart5_interrupt_flag_enum

Table 3-986. Enum usart5_interrupt_flag_enum

Member name	Function description
USART5_INT_FLAG_EB	end of block interrupt and flag
USART5_INT_FLAG_RT	receiver timeout interrupt and flag
USART5_INT_FLAG_AM	address match interrupt and flag
USART5_INT_FLAG_PERR	parity error interrupt and flag
USART5_INT_FLAG_TBE	transmitter buffer empty interrupt and flag
USART5_INT_FLAG_TC	transmission complete interrupt and flag
USART5_INT_FLAG_RBNE	read data buffer not empty interrupt and flag
USART5_INT_FLAG_RBNE_ORE RR	read data buffer not empty interrupt and overrun error flag
USART5_INT_FLAG_IDLE	IDLE line detected interrupt and flag
USART5_INT_FLAG_LBD	LIN break detected interrupt and flag
USART5_INT_FLAG_WU	wakeup from deep-sleep mode interrupt and flag
USART5_INT_FLAG_ERR_NERR	error interrupt and noise error flag
USART5_INT_FLAG_ERR_ORER	error interrupt and overrun error

Member name	Function description
R	
USART5_INT_FLAG_ERR_FERR	error interrupt and frame error flag
USART5_INT_FLAG_RFF	receive FIFO full interrupt and flag

Enum `uart_interrupt_enum`

Table 3-987. `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	interrupt enable bit of end of block event
USART_INT_RT	interrupt enable bit of receive timeout event
USART_INT_CD	collision detected interrupt

Enum `uart5_interrupt_enum`

Table 3-988. `Enum usart5_interrupt_enum`

Member name	Function description
USART5_INT_EB	end of block interrupt
USART5_INT_RT	receiver timeout interrupt
USART5_INT_AM	address match interrupt
USART5_INT_PERR	parity error interrupt
USART5_INT_TBE	transmitter buffer empty interrupt
USART5_INT_TC	transmission complete interrupt
USART5_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt
USART5_INT_IDLE	IDLE line detected interrupt
USART5_INT_LBD	LIN break detected interrupt
USART5_INT_WU	wakeup from deep-sleep mode interrupt
USART5_INT_ERR	error interrupt
USART5_INT_RFF	receive FIFO full interrupt

Enum `uart_invert_enum`

Table 3-989. `Enum usart_invert_enum`

Member name	Function description
USART_DINV_ENABLE	data bit level inversion
USART_DINV_DISABLE	data bit level not inversion

Member name	Function description
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

Enum `uart5_invert_enum`

Table 3-990. Enum `uart5_invert_enum`

Member name	Function description
USART5_DINV_ENABLE	data bit level inversion
USART5_DINV_DISABLE	data bit level not inversion
USART5_TXPIN_ENABLE	TX pin level inversion
USART5_TXPIN_DISABLE	TX pin level not inversion
USART5_RXPIN_ENABLE	RX pin level inversion
USART5_RXPIN_DISABLE	RX pin level not inversion
USART5_SWAP_ENABLE	swap TX/RX pins
USART5_SWAP_DISABLE	not swap TX/RX pins

`uart_deinit`

The description of `uart_deinit` is shown as below:

Table 3-991. Function `uart_deinit`

Function name	<code>uart_deinit</code>
Function prototype	<code>void usart_deinit(uint32_t usart_periph);</code>
Function descriptions	reset USART/UART
Precondition	-
The called functions	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	$x=0,1,2,5$
UARTx	$x=3,4$
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset USART0 */
usart_deinit (USART0);
```

uart_baudrate_set

The description of `uart_baudrate_set` is shown as below:

Table 3-992. Function `uart_baudrate_set`

Function name	uart_baudrate_set
Function prototype	<code>void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);</code>
Function descriptions	configure USART baud rate value
Precondition	-
The called functions	<code>rcu_periph_reset_enable / rcu_periph_reset_disable</code>
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	$x=0,1,2,5$
UARTx	$x=3,4$
Input parameter{in}	
baudval	baud rate value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 baud rate value */
uart_baudrate_set(USART0, 115200);
```

uart_parity_config

The description of `uart_parity_config` is shown as below:

Table 3-993. Function `uart_parity_config`

Function name	uart_parity_config
Function prototype	<code>void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);</code>
Function descriptions	configure USART parity
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	$x=0,1,2,5$
UARTx	$x=3,4$
Input parameter{in}	
paritycfg	configure USART parity
USART_PM_NONE	no parity
USART_PM_ODD	odd parity
USART_PM_EVEN	even parity

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART parity */

uart_parity_config(USART0, USART_PM EVEN);
```

uart_word_length_set

The description of `uart_word_length_set` is shown as below:

Table 3-994. Function `uart_word_length_set`

Function name	uart_word_length_set
Function prototype	void <code>uart_word_length_set(uint32_t usart_periph, uint32_t wlen);</code>
Function descriptions	configure USART word length
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	<code>x=0,1,2,5</code>
<code>UARTx</code>	<code>x=3,4</code>
Input parameter{in}	
<code>wlen</code>	USART word length
<code>USART_WL_8BIT</code>	8 bits
<code>USART_WL_9BIT</code>	9 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 word length */

uart_word_length_set(USART0, USART_WL_9BIT);
```

uart_stop_bit_set

The description of `uart_stop_bit_set` is shown as below:

Table 3-995. Function `uart_stop_bit_set`

Function name	uart_stop_bit_set
Function prototype	void <code>uart_stop_bit_set(uint32_t usart_periph, uint32_t strlen);</code>
Function descriptions	configure USART stop bit length

Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
Input parameter{in}	
stblen	USART stop bit
<i>USART_STB_1BIT</i>	1 bit
<i>USART_STB_0_5BIT</i>	0.5 bit, not available for USARTx(x=3,4)
<i>USART_STB_2BIT</i>	2 bits
<i>USART_STB_1_5BIT</i>	1.5 bits, not available for USARTx(x=3,4)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 stop bit length */
uart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

uart_enable

The description of `uart_enable` is shown as below:

Table 3-996. Function `uart_enable`

Function name	uart_enable
Function prototype	void <code>uart_enable(uint32_t usart_periph);</code>
Function descriptions	enable USART
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 */
uart_enable(USART0);
```

uart_disable

The description of `uart_disable` is shown as below:

Table 3-997. Function `uart_disable`

Function name	uart_disable
Function prototype	<code>void usart_disable(uint32_t usart_periph);</code>
Function descriptions	disable USART
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	<i>x=0,1,2,5</i>
<i>UARTx</i>	<i>x=3,4</i>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 */
usart_disable(USART0);
```

uart_transmit_config

The description of `uart_transmit_config` is shown as below:

Table 3-998. Function `uart_transmit_config`

Function name	uart_transmit_config
Function prototype	<code>void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);</code>
Function descriptions	configure USART transmitter
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	<i>x=0,1,2,5</i>
<i>UARTx</i>	<i>x=3,4</i>
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 */
uart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

uart_receive_config

The description of `uart_receive_config` is shown as below:

Table 3-999. Function `uart_receive_config`

Function name	uart_receive_config
Function prototype	void <code>uart_receive_config(uint32_t usart_periph, uint32_t rxconfig);</code>
Function descriptions	configure USART receiver
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	x=0,1,2,5
<code>UARTx</code>	x=3,4
Input parameter{in}	
<code>rxconfig</code>	enable or disable USART receiver
<code>USART_RECEIVE_ENABLE</code>	enable USART reception
<code>USART_RECEIVE_DISABLE</code>	disable USART reception
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 receiver */
uart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

uart_oversample_config

The description of `uart_oversample_config` is shown as below:

Table 3-1000. Function `uart_oversample_config`

Function name	uart_oversample_config
Function prototype	void <code>uart_oversample_config(uint32_t usart_periph, uint32_t oversamp);</code>

Function descriptions	configure the USART oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
Input parameter{in}	
oversamp	oversample value
<i>USART_OVSMOD_8</i>	8 bits
<i>USART_OVSMOD_16</i>	16 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 oversample mode */
uart_oversample_config(USART0, USART_OVSMOD_8);
```

uart_sample_bit_config

The description of `uart_sample_bit_config` is shown as below:

Table 3-1001. Function `uart_sample_bit_config`

Function name	uart_sample_bit_config
Function prototype	void <code>uart_sample_bit_config(uint32_t usart_periph, uint32_t obsm);</code>
Function descriptions	configure sample bit method
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
Input parameter{in}	
obsm	sample bit
<i>USART_OSB_1bit</i>	1 bits
<i>USART_OSB_3bit</i>	3 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure USART0 sample bit */

uart_sample_bit_config(USART0, USART_OSB_1bit);

```

uart_receiver_timeout_enable

The description of `uart_receiver_timeout_enable` is shown as below:

Table 3-1002. Function `uart_receiver_timeout_enable`

Function name	uart_receiver_timeout_enable
Function prototype	void <code>uart_receiver_timeout_enable(uint32_t usart_periph);</code>
Function descriptions	enable receiver timeout
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx peripheral
<code>USARTx</code>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable receiver timeout of USART */

uart_receiver_timeout_enable(USART0);

```

uart_receiver_timeout_disable

The description of `uart_receiver_timeout_disable` is shown as below:

Table 3-1003. Function `uart_receiver_timeout_disable`

Function name	uart_receiver_timeout_disable
Function prototype	void <code>uart_receiver_timeout_disable(uint32_t usart_periph);</code>
Function descriptions	disable receiver timeout
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx peripheral
<code>USARTx</code>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* disable receiver timeout of USART */

uart_receiver_timeout_disable(USART0);

```

uart_receiver_timeout_threshold_config

The description of `uart_receiver_timeout_threshold_config` is shown as below:

Table 3-1004. Function `uart_receiver_timeout_threshold_config`

Function name	uart_receiver_timeout_threshold_config
Function prototype	void <code>uart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);</code>
Function descriptions	configure receiver timeout threshold
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx peripheral
<code>USARTx</code>	x=0,1,2,5
Input parameter{in}	
<code>rtimeout</code>	receiver timeout threshold (0x00000000 - 0x00FFFFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* set the receiver timeout threshold of USART0 */

uart_receiver_timeout_threshold_config(USART0, 115200*3);

```

uart_data_transmit

The description of `uart_data_transmit` is shown as below:

Table 3-1005. Function `uart_data_transmit`

Function name	uart_data_transmit
Function prototype	void <code>uart_data_transmit(uint32_t usart_periph, uint32_t data);</code>
Function descriptions	USART transmit data function
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	x=0,1,2,5
<code>UARTx</code>	x=3,4
Input parameter{in}	
<code>data</code>	data of transmission

0-0x1FF	data of transmission
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 transmit data */

uart_data_transmit(USART0, 0xAA);
```

uart_data_receive

The description of `uart_data_receive` is shown as below:

Table 3-1006. Function `uart_data_receive`

Function name	uart_data_receive
Function prototype	void <code>uart_receive_config(uint32_t usart_periph, uint32_t rxconfig);</code>
Function descriptions	USART receive data function
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	x=0,1,2,5
<code>UARTx</code>	x=3,4
Output parameter{out}	
-	-
Return value	
<code>uint32_t</code>	data of received(0-0xFF)

Example:

```
/* USART0 receive data */

uint16_t temp;

temp = uart_data_receive(USART0);
```

uart_mute_mode_enable

The description of `uart_mute_mode_enable` is shown as below:

Table 3-1007. Function `uart_mute_mode_enable`

Function name	uart_mute_mode_enable
Function prototype	void <code>uart_mute_mode_enable(uint32_t usart_periph);</code>
Function descriptions	enable mute mode
Precondition	-

The called functions		-
Input parameter{in}		
uart_periph		USARTx/UARTx peripheral
USARTx		x=0,1,2,5
UARTx		x=3,4
Output parameter{out}		
-		-
Return value		
-		-

Example:

```
/* enable USART0 receiver in mute mode */

uart_mute_mode_enable(USART0);
```

uart_mute_mode_disable

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

Table 3-1008. Function `uart_mute_mode_disable`

Function name	uart_mute_mode_disable	
Function prototype	void <code>uart_mute_mode_disable(uint32_t usart_periph);</code>	
Function descriptions	disable mute mode	
Precondition	-	
The called functions	-	
Input parameter{in}		
uart_periph	USARTx/UARTx peripheral	
USARTx	x=0,1,2,5	
UARTx	x=3,4	
Output parameter{out}		
-		-
Return value		
-		-

Example:

```
/* disable USART0 receiver in mute mode */

uart_mute_mode_disable(USART0);
```

uart_mute_mode_wakeup_config

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

Table 3-1009. Function `uart_mute_mode_wakeup_config`

Function name	uart_mute_mode_wakeup_config	
Function prototype	void <code>uart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t</code>	

	wmethod);
Function descriptions	configure wakeup method in mute mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
Input parameter{in}	
wmethod	two methods be used to enter or exit the mute mode
<i>USART_WM_IDLE</i>	idle line
<i>USART_WM_ADDR</i>	address mask
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 wakeup method in mute mode */
uart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

uart_lin_mode_enable

The description of `uart_lin_mode_enable` is shown as below:

Table 3-1010. Function `uart_lin_mode_enable`

	<code>uart_lin_mode_enable</code>
Function prototype	<code>void usart_lin_mode_enable(uint32_t usart_periph);</code>
Function descriptions	enable LIN mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 LIN mode enable */
uart_lin_mode_enable(USART0);
```

uart_lin_mode_disable

The description of `uart_lin_mode_disable` is shown as below:

Table 3-1011. Function `uart_lin_mode_disable`

Function name	uart_lin_mode_disable
Function prototype	void usart_lin_mode_disable(uint32_t usart_periph);
Function descriptions	disable LIN mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 LIN mode disable */

uart_lin_mode_disable(USART0);
```

uart_lin_break_decton_length_config

The description of `uart_lin_break_decton_length_config` is shown as below:

Table 3-1012. Function `uart_lin_break_decton_length_config`

Function name	uart_lin_break_decton_length_config
Function prototype	void usart_lin_break_decton_length_config(uint32_t usart_periph, uint32_t iblen);
Function descriptions	configure LIN break frame length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
Input parameter{in}	
iblen	two methods be used to enter or exit the mute mode
<i>USART_LBLEN_10B</i>	break frame length is 10 bits
<i>USART_LBLEN_11B</i>	break frame length is 11 bits
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure LIN break frame length */
uart_lin_break_decton_length_config(USART0, USART_LBLEN_10B);
```

uart_halfduplex_enable

The description of `uart_halfduplex_enable` is shown as below:

Table 3-1013. Function `uart_halfduplex_enable`

Function name	uart_halfduplex_enable
Function prototype	void usart_halfduplex_enable(uint32_t usart_periph);
Function descriptions	enable half duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 half duplex mode*/
uart_halfduplex_enable(USART0);
```

uart_halfduplex_disable

The description of `uart_halfduplex_disable` is shown as below:

Table 3-1014. Function `uart_halfduplex_disable`

Function name	uart_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	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4

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 3-1015. 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	USARTx peripheral
USARTx	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 3-1016. 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	USARTx peripheral
USARTx	x=0,1,2,5

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 CK pin in synchronous mode */
uart_synchronous_clock_disable(USART0);
```

uart_synchronous_clock_config

The description of `uart_synchronous_clock_config` is shown as below:

Table 3-1017. Function `uart_synchronous_clock_config`

Function name	uart_synchronous_clock_config
Function prototype	void <code>uart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);</code>
Function descriptions	configure USART synchronous mode parameters
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx peripheral
<code>USARTx</code>	x=0,1,2,5
Input parameter{in}	
<code>clen</code>	CK length
<code>USART_CLEN_NONE</code>	there are 7 CK pulses for an 8 bit frame and 8 CK pulses for a 9 bit frame
<code>USART_CLEN_EN</code>	there are 8 CK pulses for an 8 bit frame and 9 CK pulses for a 9 bit frame
Input parameter{in}	
<code>cph</code>	clock phase
<code>USART_CPH_1CK</code>	first clock transition is the first data capture edge
<code>USART_CPH_2CK</code>	second clock transition is the first data capture edge
Input parameter{in}	
<code>cpl</code>	clock polarity
<code>USART_CPL_LOW</code>	steady low value on CK pin
<code>USART_CPL_HIGH</code>	steady high value on CK pin
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 synchronous mode parameters */
uart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,
```

```
USART_CPL_HIGH);
```

uart_guard_time_config

The description of `uart_guard_time_config` is shown as below:

Table 3-1018. Function `uart_guard_time_config`

Function name	uart_guard_time_config
Function prototype	void <code>uart_guard_time_config(uint32_t usart_periph,uint32_t gaut);</code>
Function descriptions	configure guard time value in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx peripheral
<code>USARTx</code>	x=0,1,2,5
Input parameter{in}	
<code>guat</code>	guard time value (0x00000000 - 0x000000FF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 guard time value in smartcard mode */
uart_guard_time_config(USART0, 0x0000 0055);
```

uart_smartcard_mode_enable

The description of `uart_smartcard_mode_enable` is shown as below:

Table 3-1019. Function `uart_smartcard_mode_enable`

Function name	uart_smartcard_mode_enable
Function prototype	void <code>uart_smartcard_mode_enable(uint32_t usart_periph);</code>
Function descriptions	enable smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx peripheral
<code>USARTx</code>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* USART0 smartcard mode enable */

uart_smartcard_mode_enable(USART0);

```

uart_smartcard_mode_disable

The description of `uart_smartcard_mode_disable` is shown as below:

Table 3-1020. Function `uart_smartcard_mode_disable`

Function name	uart_smartcard_mode_disable
Function prototype	void <code>uart_smartcard_mode_disable(uint32_t usart_periph);</code>
Function descriptions	disable smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* USART0 smartcard mode disable */

uart_smartcard_mode_disable(USART0);

```

uart_smartcard_mode_nack_enable

The description of `uart_smartcard_mode_nack_enable` is shown as below:

Table 3-1021. Function `uart_smartcard_mode_nack_enable`

Function name	uart_smartcard_mode_nack_enable
Function prototype	void <code>uart_smartcard_mode_nack_enable(uint32_t usart_periph);</code>
Function descriptions	enable NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 NACK in smartcard mode */
```

```
uart_smartcard_mode_nack_enable(USART0);
```

uart_smartcard_mode_nack_disable

The description of `uart_smartcard_mode_nack_disable` is shown as below:

Table 3-1022. Function `uart_smartcard_mode_nack_disable`

Function name	uart_smartcard_mode_nack_disable
Function prototype	void <code>uart_smartcard_mode_nack_disable(uint32_t usart_periph);</code>
Function descriptions	disable NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx peripheral
<code>USARTx</code>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 NACK in smartcard mode */
```

```
uart_smartcard_mode_nack_disable(USART0);
```

uart_smartcard_autoretry_config

The description of `uart_smartcard_autoretry_config` is shown as below:

Table 3-1023. Function `uart_smartcard_autoretry_config`

Function name	uart_smartcard_autoretry_config
Function prototype	void <code>uart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);</code>
Function descriptions	configure smartcard auto-retry number
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx peripheral
<code>USARTx</code>	x=0,1,2,5
Input parameter{in}	
<code>scrtnum</code>	smartcard auto-retry number (0x00000000 - 0x00000007)
Output parameter{out}	
-	-
Return value	

Example:

```
/* configure smartcard auto-retry number */

uart_smartcard_autoretry_config (USART0, 0x00000007);
```

uart_block_length_config

The description of `uart_block_length_config` is shown as below:

Table 3-1024. Function `uart_block_length_config`

Function name	uart_block_length_config
Function prototype	void <code>uart_block_length_config(uint32_t usart_periph, uint32_t bl);</code>
Function descriptions	configure block length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
bl	block length (0x00000000 - 0x000000FF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure block length in Smartcard T=1 reception */

uart_block_length_config(USART0, 0x000000FF);
```

uart_irda_mode_enable

The description of `uart_irda_mode_enable` is shown as below:

Table 3-1025. Function `uart_irda_mode_enable`

Function name	uart_irda_mode_enable
Function prototype	void <code>uart_irda_mode_enable(uint32_t usart_periph);</code>
Function descriptions	enable IrDA mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 IrDA mode */
uart_irda_mode_enable(USART0);
```

uart_irda_mode_disable

The description of `uart_irda_mode_disable` is shown as below:

Table 3-1026. Function `uart_irda_mode_disable`

Function name	uart_irda_mode_disable
Function prototype	void <code>uart_irda_mode_disable(uint32_t usart_periph);</code>
Function descriptions	disable IrDA mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 IrDA mode */
uart_irda_mode_disable(USART0);
```

uart_prescaler_config

The description of `uart_prescaler_config` is shown as below:

Table 3-1027. Function `uart_prescaler_config`

Function name	uart_prescaler_config
Function prototype	void <code>uart_prescaler_config(uint32_t usart_periph, uint8_t psc);</code>
Function descriptions	configure the peripheral clock prescaler in USART IrDA low-power mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral

USARTx	x=0,1,2,5
UARTx	x=3,4
Input parameter{in}	
psc	0x00000000 - 0x000000FF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the USART0 peripheral clock prescaler */
uart_prescaler_config(USART0, 0x00000000);
```

uart_irda_lowpower_config

The description of `uart_irda_lowpower_config` is shown as below:

Table 3-1028. Function `uart_irda_lowpower_config`

Function name	uart_irda_lowpower_config
Function prototype	void <code>uart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);</code>
Function descriptions	configure IrDA low-power
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4
Input parameter{in}	
<code>irlp</code>	IrDA low-power or normal
<code>USART_IRLP_LOW</code>	low-power
<code>USART_IRLP_NORMAL</code>	normal
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 IrDA low-power */
uart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

uart_dma_receive_config

The description of usart_dma_receive_config is shown as below:

Table 3-1029. Function usart_dma_receive_config

Function name	usart_dma_receive_config
Function prototype	void usart_dma_receive_config(uint32_t usart_periph, uint8_t dmacmd);
Function descriptions	configure USART DMA reception
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
Input parameter{in}	
dmacmd	enable or disable DMA for reception
<i>USART_RECEIVE_DM_A_ENABLE</i>	DMA enable for reception
<i>USART_RECEIVE_DM_A_DISABLE</i>	DMA disable for reception
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 DMA enable for reception */
usart_dma_receive_config(USART0, USART_DENR_ENABLE);
```

uart_dma_transmit_config

The description of usart_dma_transmit_config is shown as below:

Table 3-1030. Function usart_dma_transmit_config

Function name	usart_dma_transmit_config
Function prototype	void usart_dma_transmit_config(uint32_t usart_periph, uint8_t dmacmd);
Function descriptions	configure USART DMA transmission
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4
Input parameter{in}	

dmacmd	enable or disable DMA for transmission
USART_TRANSMIT_DMA_ENABLE	DMA enable for transmission
USART_TRANSMIT_DMA_DISABLE	DMA disable for transmission
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 DMA enable for transmission */

uart_dma_transmit_config(USART0, USART_DENT_ENABLE);
```

uart_hardware_flow_rts_config

The description of `uart_hardware_flow_rts_config` is shown as below:

Table 3-1031. Function `uart_hardware_flow_rts_config`

Function name	uart_hardware_flow_rts_config
Function prototype	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
Function descriptions	configure hardware flow control RTS
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2
Input parameter{in}	
rtsconfig	enable or disable RTS
USART_RTS_ENABLE	enable RTS
USART_RTS_DISABLE	disable RTS
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 hardware flow control RTS */

uart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

uart_hardware_flow_cts_config

The description of usart_hardware_flow_cts_config is shown as below:

Table 3-1032. Function usart_hardware_flow_cts_config

Function name	usart_hardware_flow_cts_config
Function prototype	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
Function descriptions	configure hardware flow control RTS
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2
Input parameter{in}	
ctsconfig	enable or disable CTS
USART_CTS_ENABLE	enable CTS
USART_CTS_DISABLE	disable CTS
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 hardware flow control CTS */
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

uart_data_first_config

The description of usart_data_first_config is shown as below:

Table 3-1033. 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	USARTx peripheral
USARTx	x=0,1,2
Input parameter{in}	
msbf	LSB first or MSB first
USART_MSBF_LSB	LSB first

USART_MSBF_MSB	MSB first
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure LSB of data first */

uart_data_first_config(USART0, USART_MSBF_LSB);
```

uart_invert_config

The description of `uart_invert_config` is shown as below:

Table 3-1034. Function `uart_invert_config`

Function name	uart_invert_config
Function prototype	void <code>uart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);</code>
Function descriptions	configure USART inversion
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx peripheral
<code>USARTx</code>	x=0,1,2
Input parameter{in}	
<code>invertpara</code>	refer to Table 3-989. Enum <code>usart_invert_enum</code>
<code>USART_DINV_ENABL_E</code>	data bit level inversion
<code>USART_DINV_DISABL_E</code>	data bit level not inversion
<code>USART_TXPIN_ENABL_E</code>	TX pin level inversion
<code>USART_TXPIN_DISABL_E</code>	TX pin level not inversion
<code>USART_RXPIN_ENABL_E</code>	RX pin level inversion
<code>USART_RXPIN_DISABL_E</code>	RX pin level not inversion
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure USART inversion */

uart_invert_config(USART0, USART_DINV_ENABLE);

```

uart_address_config

The description of `uart_address_config` is shown as below:

Table 3-1035. Function `uart_address_config`

Function name	uart_address_config
Function prototype	void <code>uart_address_config(uint32_t usart_periph, uint8_t addr)</code> ;
Function descriptions	configure the address of the USART in wake up by address match mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	x=0,1,2
<code>UARTx</code>	x=3,4
Input parameter{in}	
<code>addr</code>	address of USART/UART (0x00000000 - 0x0000000F)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure address of the USART0 */

uart_address_config(USART0, 0x00000000);

```

uart_send_break

The description of `uart_send_break` is shown as below:

Table 3-1036. Function `uart_send_break`

Function name	uart_send_break
Function prototype	void <code>uart_send_break(uint32_t usart_periph)</code> ;
Function descriptions	send break frame
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	x=0,1,2
<code>UARTx</code>	x=3,4
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* USART0 send break frame */

uart_send_break(USART0);
```

uart_flag_get

The description of `uart_flag_get` is shown as below:

Table 3-1037. Function `uart_flag_get`

Function name	uart_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	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Input parameter{in}	
flag	USART flags, refer to Table 3-983. Enum usart_flag_enum
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 frame detected flag
USART_FLAG_ORER <i>R</i>	overrun error flag
USART_FLAG_NERR	noise error flag
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
USART_FLAG_CD	collision detected flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```

/* get flag USART0 state */

FlagStatus status;

status = usart_flag_get(USART0, USART_FLAG_TBE);
  
```

usart_flag_clear

The description of usart_flag_clear is shown as below:

Table 3-1038. 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	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Input parameter{in}	
flag	USART flags, refer to Table 3-983. 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
USART_FLAG_CD	collision detected 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 3-1039. 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}	
uart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Input parameter{in}	
interrupt	USART interrupt, refer to Table 3-987. Enum usart_interrupt_enum
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_ERR	error interrupt
USART_INT_CTS	CTS interrupt
USART_INT_RT	receive timeout event interrupt
USART_INT_EB	end of block event interrupt
USART_INT_CD	collision detected interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 TBE interrupt */

uart_interrupt_enable(USART0, USART_INT_TBE);
```

uart_interrupt_disable

The description of `uart_interrupt_disable` is shown as below:

Table 3-1040. Function `uart_interrupt_disable`

Function name	uart_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}	
uart_periph	USARTx/UARTx peripheral

USARTx	x=0,1,2
UARTx	x=3,4
Input parameter{in}	
int_flag	USART interrupt flag, refer to Table 3-987. Enum usart_interrupt_enum
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_ERR	error interrupt
USART_INT_CTS	CTS interrupt
USART_INT_RT	receive timeout event interrupt
USART_INT_EB	end of block event interrupt
USART_INT_CD	collision detected interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 TBE interrupt */

uart_interrupt_disable(USART0, USART_INT_TBE);
```

uart_interrupt_flag_get

The description of `uart_interrupt_flag_get` is shown as below:

Table 3-1041. Function `uart_interrupt_flag_get`

Function name	<code>uart_interrupt_flag_get</code>
Function prototype	<code>FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);</code>
Function descriptions	get USART interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Input parameter{in}	
int_flag	USART interrupt flag, refer to Table 3-985. Enum usart_interrupt_flag_enum
USART_INT_FLAG_PE	parity error interrupt and flag

RR	
<i>USART_INT_FLAG_TB E</i>	transmitter buffer empty interrupt and flag
<i>USART_INT_FLAG_TC</i>	transmission complete interrupt and flag
<i>USART_INT_FLAG_RB NE</i>	read data buffer not empty interrupt and flag
<i>USART_INT_FLAG_RB NE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>USART_INT_FLAG_ID LE</i>	IDLE line detected interrupt and flag
<i>USART_INT_FLAG_LB D</i>	LIN break detected interrupt and flag
<i>USART_INT_FLAG_CT S</i>	CTS interrupt and flag
<i>USART_INT_FLAG_ER R_ORERR</i>	error interrupt and overrun error
<i>USART_INT_FLAG_ER R_NERR</i>	error interrupt and noise error flag
<i>USART_INT_FLAG_ER R_FERR</i>	error interrupt and frame error flag
<i>USART_INT_FLAG_EB</i>	end of block event interrupt flag
<i>USART_INT_FLAG_RT</i>	receive timeout event interrupt flag
<i>USART_INT_FLAG_C D</i>	collision detected interrupt and flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the USART0 interrupt flag status */

FlagStatus status;

status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
```

usart_interrupt_flag_clear

The description of usart_interrupt_flag_clear is shown as below:

Table 3-1042. 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

Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
Input parameter{in}	
int_flag	USART interrupt flag, refer to Table 3-985. Enum usart_interrupt_flag_enum
<i>USART_INT_FLAG_CT_S</i>	CTS interrupt and flag
<i>USART_INT_FLAG_LB_D</i>	LIN break detected interrupt and flag
<i>USART_INT_FLAG_TC</i>	transmission complete interrupt and flag
<i>USART_INT_FLAG_RB_{NE}</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
<i>USART_INT_FLAG_C_D</i>	collision detected interrupt and flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the USART0 interrupt enable bit status */
uart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

uart5_data_first_config

The description of `uart_data_first_config` is shown as below:

Table 3-1043. Function `uart5_data_first_config`

Function name	uart5_data_first_config
Function prototype	void <code>uart5_data_first_config(uint32_t usart_periph, uint32_t msbf);</code>
Function descriptions	data is transmitted/received with the LSB/MSB first
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	uart peripheral
<i>USARTx</i>	x=5
Input parameter{in}	

msbf	LSB/MSB
USART5_MSBF_LSB	LSB first
USART5_MSBF_MSB	MSB first
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure LSB of data first */

uart5_data_first_config(USART5, USART5_MSBF_LSB);
```

uart5_invert_config

The description of `uart5_invert_config` is shown as below:

Table 3-1044. Function `uart5_invert_config`

Function name	<code>uart5_invert_config</code>
Function prototype	<code>void usart5_invert_config(uint32_t usart_periph, usart5_invert_enum invertpara);</code>
Function descriptions	configure USART5 inverted
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	uart peripheral
USARTx	x=5
Input parameter{in}	
invertpara	refer to Table 3-990. Enum <code>usart5_invert_enum</code>
USART5_DINV_ENAB LE	data bit level inversion
USART5_DINV_DISAB LE	data bit level not inversion
USART5_TXPIN_ENA BLE	TX pin level inversion
USART5_TXPIN_DISA BLE	TX pin level not inversion
USART5_RXPIN_ENA BLE	RX pin level inversion
USART5_RXPIN_DISA BLE	RX pin level not inversion
USART5_SWAP_ENA BLE	swap TX/RX pins
USART5_SWAP_DISA	not swap TX/RX pins

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

Example:

```
/* configure USART5 inversion */

uart5_invert_config(USART5, USART5_DINV_ENABLE);
```

uart5_overrun_enable

The description of `uart5_overrun_enable` is shown as below:

Table 3-1045. Function `uart5_overrun_enable`

Function name	uart5_overrun_enable
Function prototype	void uart5_overrun_enable(uint32_t usart_periph);
Function descriptions	enable the USART5 overrun function
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	uart peripheral
USARTx	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART5 overrun */

uart5_overrun_enable(USART5);
```

uart5_overrun_disable

The description of `uart5_overrun_disable` is shown as below:

Table 3-1046. Function `uart5_overrun_disable`

Function name	uart5_overrun_disable
Function prototype	void uart5_overrun_disable(uint32_t usart_periph);
Function descriptions	disable the USART5 overrun function
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	uart peripheral

USARTx	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART5 overrun */

uart5_overrun_disable(USART5);
```

uart5_address_config

The description of `uart5_address_config` is shown as below:

Table 3-1047. Function `uart5_address_config`

Function name	uart5_address_config
Function prototype	void uart5_address_config(uint32_t usart_periph, uint8_t addr);
Function descriptions	configure the address of the USART5 terminal
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	uart peripheral
USARTx	x=5
Input parameter{in}	
addr	address of USART terminal (0x00000000-0x000000FF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure address of the USART5 */

uart5_address_config(USART5, 0x00000000);
```

uart5_address_detection_mode_config

The description of `uart5_address_detection_mode_config` is shown as below:

Table 3-1048. Function `uart5_address_detection_mode_config`

Function name	uart5_address_detection_mode_config
Function prototype	void uart5_address_detection_mode_config(uint32_t usart_periph, uint32_t addmod);
Function descriptions	configure address detection mode
Precondition	-

The called functions		-
Input parameter{in}		
uart_periph		uart peripheral
USARTx		x=5
Input parameter{in}		
addmod		address detection mode
USART5_ADDM_4BIT		4 bits
USART5_ADDM_FULL BIT		full bits
Output parameter{out}		
-	-	-
Return value		
-	-	-

Example:

```
/*configure address detection */
uart5_address_detection_mode_config(USART5, USART5_ADDM_4BIT);
```

uart5_smartcard_mode_early_nack_enable

The description of **uart5_smartcard_mode_early_nack_enable** is shown as below:

Table 3-1049. Function `uart5_smartcard_mode_early_nack_enable`

Function name		uart5_smartcard_mode_early_nack_enable
Function prototype		void uart5_smartcard_mode_early_nack_enable(uint32_t usart_periph);
Function descriptions		enable early NACK in smartcard mode
Precondition		-
The called functions		-
Input parameter{in}		
uart_periph		uart peripheral
USARTx		x=5
Output parameter{out}		
-	-	-
Return value		
-	-	-

Example:

```
/* enable USART5 early NACK in smartcard mode */
uart5_smartcard_mode_early_nack_enable(USART5);
```

uart5_smartcard_mode_early_nack_disable

The description of **uart5_smartcard_mode_early_nack_disable** is shown as below:

Table 3-1050. Function `uart5_smartcard_mode_early_nack_disable`

Function name	uart5_smartcard_mode_early_nack_disable
Function prototype	void usart5_smartcard_mode_early_nack_disable(uint32_t usart_periph);
Function descriptions	disable early NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART5 early NACK in smartcard mode */

uart5_smartcard_mode_early_nack_disable(USART5);
```

`uart5_reception_error_dma_enable`

The description of `uart5_reception_error_dma_enable` is shown as below:

 Table 3-1051. Function `uart5_reception_error_dma_enable`

Function name	uart5_reception_error_dma_enable
Function prototype	void usart5_reception_error_dma_enable(uint32_t usart_periph);
Function descriptions	enable DMA on reception error
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA on reception error */

uart5_reception_error_dma_enable(USART5);
```

`uart5_reception_error_dma_disable`

The description of `uart5_reception_error_dma_disable` is shown as below:

Table 3-1052. Function **uart5_reception_error_dma_disable**

Function name	uart5_reception_error_dma_disable
Function prototype	void usart5_reception_error_dma_disable(uint32_t usart_periph);
Function descriptions	disable DMA on reception error
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	uart peripheral
USARTx	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA on reception error */

uart5_reception_error_dma_disable(USART5);
```

uart5_wakeup_enable

The description of **uart5_wakeup_enable** is shown as below:

 Table 3-1053. Function **uart5_wakeup_enable**

Function name	uart5_wakeup_enable
Function prototype	void usart5_wakeup_enable(uint32_t usart_periph);
Function descriptions	enable USART to wakeup the mcu from deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	uart peripheral
USARTx	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART5 wake up enable */

uart5_wakeup_enable(USART5);
```

uart5_wakeup_disable

The description of **uart5_wakeup_disable** is shown as below:

Table 3-1054. Function `uart5_wakeup_disable`

Function name	uart5_wakeup_disable
Function prototype	void usart5_wakeup_disable(uint32_t usart_periph);
Function descriptions	disable USART to wakeup the mcu from deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART5 wake up disable */

uart5_wakeup_disable(USART5);
```

`uart5_wakeup_mode_config`

The description of `uart5_wakeup_mode_config` is shown as below:

 Table 3-1055. Function `uart5_wakeup_mode_config`

Function name	uart5_wakeup_mode_config
Function prototype	void usart5_wakeup_mode_config(uint32_t usart_periph, uint32_t wum);
Function descriptions	configure the USART wakeup mode from deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=5
Input parameter{in}	
<code>wum</code>	wakeup mode
<code>USART5_WUM_ADDR</code>	WUF active on address match
<code>USART5_WUM_STAR_TB</code>	WUF active on start bit
<code>USART5_WUM_RBNE</code>	WUF active on RBNE
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure USART5 wake up mode */

uart5_wakeup_mode_config(USART5, USART5_WUM_ADDR);

```

uart5_receive_fifo_enable

The description of `uart5_receive_fifo_enable` is shown as below:

Table 3-1056. Function `uart5_receive_fifo_enable`

Function name	uart5_receive_fifo_enable
Function prototype	void <code>uart5_receive_fifo_enable(uint32_t usart_periph);</code>
Function descriptions	enable receive FIFO
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable receive FIFO */

uart5_receive_fifo_enable(USART5);

```

uart5_receive_fifo_disable

The description of `uart5_receive_fifo_disable` is shown as below:

Table 3-1057. Function `uart5_receive_fifo_disable`

Function name	Usart5_receive_fifo_disable
Function prototype	void <code>uart5_receive_fifo_disable(uint32_t usart_periph);</code>
Function descriptions	disable receive FIFO
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* disable receive FIFO */

uart5_receive_fifo_disable(USART5);

```

uart5_receive_fifo_counter_number

The description of `uart5_receive_fifo_counter_number` is shown as below:

Table 3-1058. Function `uart5_receive_fifo_counter_number`

Function name	uart5_receive_fifo_counter_number
Function prototype	uint8_t <code>uart5_receive_fifo_counter_number(uint32_t usart_periph);</code>
Function descriptions	read receive FIFO counter number
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=5
Output parameter{out}	
-	-
Return value	
<code>uint8_t</code>	receive FIFO counter number

Example:

```

/* read receive FIFO counter number */

uint8_t temp;

temp = uart5_receive_fifo_counter_number(USART5);

```

uart5_flag_get

The description of `uart5_flag_get` is shown as below:

Table 3-1059. Function `uart5_flag_get`

Function name	uart5_flag_get
Function prototype	FlagStatus <code>uart5_flag_get(uint32_t usart_periph, usart5_flag_enum flag);</code>
Function descriptions	get flag in STAT/CHC/RFCS register
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=5
Input parameter{in}	
<code>flag</code>	USART flags, refer to Table 3-984. Enum <code>uart5_flag_enum</code> only one among these parameters can be selected
<code>USART5_FLAG_PERR</code>	parity error flag

<code>USART5_FLAG_FERR</code>	frame error flag
<code>USART5_FLAG_NERR</code>	noise error flag
<code>USART5_FLAG_ORER</code> <i>R</i>	overrun error
<code>USART5_FLAG_IDLE</code>	idle line detected flag
<code>USART5_FLAG_RBNE</code>	read data buffer not empty
<code>USART5_FLAG_TC</code>	transmission completed
<code>USART5_FLAG_TBE</code>	transmit data register empty
<code>USART5_FLAG_LBD</code>	LIN break detected flag
<code>USART5_FLAG_RT</code>	receiver timeout flag
<code>USART5_FLAG_EB</code>	end of block flag
<code>USART5_FLAG_BSY</code>	busy flag
<code>USART5_FLAG_AM</code>	address match flag
<code>USART5_FLAG_SB</code>	send break flag
<code>USART5_FLAG_RWU</code>	receiver wakeup from mute mode
<code>USART5_FLAG_WU</code>	wakeup from deep-sleep mode flag
<code>USART5_FLAG_TEA</code>	transmit enable acknowledge flag
<code>USART5_FLAG_REA</code>	receive enable acknowledge flag
<code>USART5_FLAG_EPER</code> <i>R</i>	early parity error flag
<code>USART5_FLAG_RFE</code>	receive FIFO empty flag
<code>USART5_FLAG_RFF</code>	receive FIFO full flag
<code>USART5_FLAG_RFFIN</code> <i>T</i>	receive FIFO full interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get flag USART5 state */

FlagStatus status;

status = usart5_flag_get(USART5, USART5_FLAG_TBE);
```

usart5_flag_clear

The description of usart5_flag_clear is shown as below:

Table 3-1060. Function usart5_flag_clear

Function name	usart5_flag_clear
----------------------	-------------------

Function prototype	void usart5_flag_clear(uint32_t usart_periph, usart5_flag_enum flag);
Function descriptions	clear flag in STAT register
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	uart peripheral
USARTx	x=5
Input parameter{in}	
flag	USART flags, refer to Table 3-984. Enum usart5_flag_enum only one among these parameters can be selected
USART5_FLAG_PERR	parity error flag
USART5_FLAG_FERR	frame error flag
USART5_FLAG_NERR	noise detected flag
USART5_FLAG_ORER <i>R</i>	overrun error flag
USART5_FLAG_IDLE	idle line detected flag
USART5_FLAG_TC	transmission complete flag
USART5_FLAG_LBD	LIN break detected flag
USART5_FLAG_RT	receiver timeout flag
USART5_FLAG_EB	end of block flag
USART5_FLAG_AM	address match flag
USART5_FLAG_WU	wakeup from deep-sleep mode flag
USART5_FLAG_EPER <i>R</i>	early parity error flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear USART5 flag */

usart5_flag_clear(USART5, USART5_FLAG_TC);
```

usart5_interrupt_enable

The description of usart5_interrupt_enable is shown as below:

Table 3-1061. Function usart5_interrupt_enable

Function name	usart5_interrupt_enable
Function prototype	void usart5_interrupt_enable(uint32_t usart_periph, usart5_interrupt_enum interrupt);
Function descriptions	enable USART interrupt
Precondition	-

The called functions		-
Input parameter{in}		
uart_periph	uart peripheral	
USARTx	x=5	
Input parameter{in}		
interrupt	USART5 interrupts, refer to Table 3-988. Enum usart5_interrupt enum	
USART5_INT_IDLE	idle interrupt	
USART5_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt enable interrupt	
USART5_INT_TC	transmission complete interrupt	
USART5_INT_TBE	transmit data register empty interrupt	
USART5_INT_PERR	parity error interrupt	
USART5_INT_AM	address match interrupt	
USART5_INT_RT	receiver timeout interrupt	
USART5_INT_EB	end of block interrupt	
USART5_INT_LBD	LIN break detection interrupt	
USART5_INT_ERR	error interrupt enable in multibuffer communication	
USART5_INT_WU	wakeup from deep-sleep mode interrupt	
USART5_INT_RFF	receive FIFO full interrupt enable	
Output parameter{out}		
-	-	
Return value		
-	-	

Example:

```
/* enable USART5 TBE interrupt */
uart5_interrupt_enable(USART5, USART5_INT_TBE);
```

uart5_interrupt_disable

The description of `uart5_interrupt_disable` is shown as below:

Table 3-1062. Function `uart5_interrupt_disable`

Function name	uart5_interrupt_disable	
Function prototype	void usart5_interrupt_disable(uint32_t usart_periph, usart5_interrupt_enum interrupt);	
Function descriptions	disable USART interrupt	
Precondition	-	
The called functions	-	
Input parameter{in}		
uart_periph	uart peripheral	
USARTx	x=5	
Input parameter{in}		

interrupt	USART5 interrupts, refer to Table 3-988. Enum usart5_interrupt_enum
USART5_INT_IDLE	idle interrupt
USART5_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt enable interrupt
USART5_INT_TC	transmission complete interrupt
USART5_INT_TBE	transmit data register empty interrupt
USART5_INT_PERR	parity error interrupt
USART5_INT_AM	address match interrupt
USART5_INT_RT	receiver timeout interrupt
USART5_INT_EB	end of block interrupt
USART5_INT_LBD	LIN break detection interrupt
USART5_INT_ERR	error interrupt enable in multibuffer communication
USART5_INT_WU	wakeup from deep-sleep mode interrupt
USART5_INT_RFF	receive FIFO full interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART5 TBE interrupt */

uart5_interrupt_disable(USART5, USART5_INT_TBE);
```

uart5_command_enable

The description of `uart5_command_enable` is shown as below:

Table 3-1063. Function `uart5_command_enable`

Function name	uart5_command_enable
Function prototype	void <code>uart5_command_enable(uint32_t usart_periph, uint32_t cmdtype);</code>
Function descriptions	enable USART command
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	uart peripheral
USARTx	x=5
Input parameter{in}	
cmdtype	command type
USART5_CMD_SBKC MD	send break command
USART5_CMD_MMCM D	mute mode command
USART5_CMD_RXFC	receive data flush command

MD	
USART5_CMD_TXFC	transmit data flush request
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART5 command */

uart5_command_enable(USART5, USART5_CMD_SBKCMD);
```

uart5_interrupt_flag_get

The description of `uart5_interrupt_flag_get` is shown as below:

Table 3-1064. Function `uart5_interrupt_flag_get`

Function name	uart5_interrupt_flag_get
Function prototype	FlagStatus usart5_interrupt_flag_get(uint32_t usart_periph, usart5_interrupt_flag_enum int_flag);
Function descriptions	get USART interrupt and flag status
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	uart peripheral
USARTx	x=5
Input parameter{in}	
int_flag	USART interrupt flag, refer to Table 3-986. Enum usart5_interrupt_flag_enum
USART5_INT_FLAG_E B	end of block interrupt and flag
USART5_INT_FLAG_R T	receiver timeout interrupt and flag
USART5_INT_FLAG_A M	address match interrupt and flag
USART5_INT_FLAG_P ERR	parity error interrupt and flag
USART5_INT_FLAG_T BE	transmitter buffer empty interrupt and flag
USART5_INT_FLAG_T C	transmission complete interrupt and flag
USART5_INT_FLAG_R BNE	read data buffer not empty interrupt and flag

<i>USART5_INT_FLAG_R</i> <i>BNE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>USART5_INT_FLAG_I</i> <i>DLE</i>	IDLE line detected interrupt and flag
<i>USART5_INT_FLAG_L</i> <i>BD</i>	LIN break detected interrupt and flag
<i>USART5_INT_FLAG_WU</i>	wakeup from deep-sleep mode interrupt and flag
<i>USART5_INT_FLAG_E</i> <i>RR_NERR</i>	error interrupt and noise error flag
<i>USART5_INT_FLAG_E</i> <i>RR_ORERR</i>	error interrupt and overrun error
<i>USART5_INT_FLAG_E</i> <i>RR_FERR</i>	error interrupt and frame error flag
<i>USART5_INT_FLAG_R</i> <i>FF</i>	receive FIFO full interrupt and flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the USART5 interrupt flag status */

FlagStatus status;

status = usart5_interrupt_flag_get(USART5, USART5_INT_FLAG_RBNE);
```

usart5_interrupt_flag_clear

The description of usart5_interrupt_flag_clear is shown as below:

Table 3-1065. Function usart5_interrupt_flag_clear

Function name	usart5_interrupt_flag_clear
Function prototype	void usart5_interrupt_flag_clear(uint32_t usart_periph, usart5_interrupt_flag_enum flag);
Function descriptions	clear USART interrupt flag in STAT register
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	usart peripheral
<i>USARTx</i>	x=5
Input parameter{in}	
flag	USART interrupt flag, refer to Table 3-986. Enum usart5_interrupt_flag_enum

<i>USART5_INT_FLAG_P</i> <i>ERR</i>	parity error interrupt and flag
<i>USART5_INT_FLAG_E</i> <i>RR_FERR</i>	error interrupt and frame error flag
<i>USART5_INT_FLAG_E</i> <i>RR_NERR</i>	error interrupt and noise error flag
<i>USART5_INT_FLAG_R</i> <i>BNE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>USART5_INT_FLAG_E</i> <i>RR_ORERR</i>	error interrupt and overrun error
<i>USART5_INT_FLAG_I</i> <i>DLE</i>	IDLE line detected interrupt and flag
<i>USART5_INT_FLAG_T</i> <i>C</i>	transmission complete interrupt and flag
<i>USART5_INT_FLAG_L</i> <i>BD</i>	LIN break detected interrupt and flag
<i>USART5_INT_FLAG_R</i> <i>T</i>	receiver timeout interrupt and flag
<i>USART5_INT_FLAG_E</i> <i>B</i>	end of block interrupt and flag
<i>USART5_INT_FLAG_A</i> <i>M</i>	address match interrupt and flag
<i>USART5_INT_FLAG_WU</i>	wakeup from deep-sleep mode interrupt and flag
<i>USART5_INT_FLAG_R</i> <i>FF</i>	receive FIFO full interrupt and flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the USART5 interrupt flag */
uart5_interrupt_flag_clear(USART5, USART5_INT_FLAG_TC);
```

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 FWDGT 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 3-1066. 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 3-1067. 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_interrupt_enable	enable early wakeup interrupt of WWDGT
wwdgt_flag_get	check early wakeup interrupt state of WWDGT
wwdgt_flag_clear	clear early wakeup interrupt state of WWDGT

wwdgt_deinit

The description of wwdgt_deinit is shown as below:

Table 3-1068. Function wwdgt_deinit

Function name	wwdgt_deinit
Function prototype	void wwdgt_deinit(void);
Function descriptions	reset the window watchdog timer configuration
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the window watchdog timer configuration */
```

```
wwdgt_deinit ( );
```

wwdgt_enable

The description of wwdgt_enable is shown as below:

Table 3-1069. 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 3-1070. Function wwdgt_counter_update

Function name	wwdgt_counter_update
Function prototype	void wwdgt_counter_update(uint16_t counter_value);
Function descriptions	configure the WWDGT 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 3-1071. 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
WWDGT_CFG_PSC_DIV1	the time base of window watchdog counter = (PCLK1/4096)/1
WWDGT_CFG_PSC_DIV2	the time base of window watchdog counter = (PCLK1/4096)/2
WWDGT_CFG_PSC_DIV4	the time base of window watchdog counter = (PCLK1/4096)/4
WWDGT_CFG_PSC_DIV8	the time base of window watchdog counter = (PCLK1/4096)/8
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* confiure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to
8 */

wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

wwdgt_interrupt_enable

The description of wwdgt_interrupt_enable is shown as below:

Table 3-1072. Function wwdgt_interrupt_enable

Function name	wwdgt_interrupt_enable
Function prototype	void wwdgt_interrupt_enable(void);
Function descriptions	enable early wakeup interrupt of WWDGT
Precondition	-
The called functions	-

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

Example:

```
/* enable early wakeup interrupt of WWDGT */
wwdgt_interrupt_enable ( );
```

wwdgt_flag_get

The description of wwdgt_flag_get is shown as below:

Table 3-1073. Function wwdgt_flag_get

Function name	wwdgt_flag_get
Function prototype	FlagStatus wwdgt_flag_get(void);
Function descriptions	check early wakeup interrupt state of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* test if the counter value update has reached the 0x40 */
FlagStatus status;

status = wwdgt_flag_get ( );

if(status == RESET)

{
  ...
}

}else

{
  ...
}
```

wwdgt_flag_clear

The description of wwdgt_flag_clear is shown as below:

Table 3-1074. Function wwdgt_flag_clear

Function name	wwdgt_flag_clear
Function prototype	void wwdgt_flag_clear(void);
Function descriptions	clear early wakeup interrupt state of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear early wakeup interrupt state of WWDGT */  
wwdgt_flag_clear();
```

4. Revision history

Table 4-1. Revision history

Revison No.	Description	Date
1.0	Initial Release	Mar.10, 2020
1.1	Module information modified	Aug.26, 2020
1.2	Module information modified	Mar.31, 2021
1.3	Module information modified	Dec.21, 2021
1.4	<ol style="list-style-type: none"> 1. Add function fwdgt_prescaler_value_config and fwdgt_reload_value_config in 3.15.2. 2. Change function i2c_dma_enable / i2c_smbus_issue_alert / i2c_smbus_arp_enable to i2c_dma_config / i2c_smbus_alert_config / i2c_smbus_arp_config in 3.18.2. 3. Change function pmu_to_standbymode(WFI_CMD) to pmu_to_standbymode() in 3.20.2. 4. Change function qspi_enable / qspi_disable / qspi_write_enable / qspi_read_enable / qspi_io23_output_enable / qspi_io23_output_disable to spi_quad_enable / spi_quad_disable / spi_quad_write_enable / spi_quad_read_enable / spi_quad_output_enable / spi_quad_io23_output_disable in 3.24.2. 5. Delete the description of GD32E50X_XD. 	Aug.08, 2022
1.5	<ol style="list-style-type: none"> 1. Modify the parameters of the function interfaces usart_dma_receive_config and usart_dma_transmit_config in 3.28.2. 	Dec.28, 2022

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company under the intellectual property laws and treaties of the People's Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as its suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as 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 Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.