

**GigaDevice Semiconductor Inc.**

**GD32W51x\_F5HC**

**Arm<sup>®</sup> Cortex<sup>®</sup>-M33 32-bit MCU**

**Firmware Library  
User Guide**

Revision 2.0

(Apr. 2026)

# Table of Contents

<b>Table of Contents .....</b>	<b>2</b>
<b>List of Figures .....</b>	<b>6</b>
<b>List of Tables .....</b>	<b>7</b>
<b>1. Introduction .....</b>	<b>34</b>
<b>1.1. Rules of User Manual and Firmware Library .....</b>	<b>34</b>
1.1.1. Peripherals.....	34
1.1.2. Naming rules.....	35
<b>2. Firmware Library Overview .....</b>	<b>37</b>
<b>2.1. File Structure of Firmware Library .....</b>	<b>37</b>
2.1.1. Examples Folder .....	38
2.1.2. Firmware Folder.....	38
2.1.3. Template Folder .....	39
2.1.4. Template_TrustZone Folder.....	41
2.1.5. Utilities Folder .....	45
<b>2.2. File descriptions of Firmware Library .....</b>	<b>45</b>
<b>3. Firmware Library of Standard Peripherals .....</b>	<b>46</b>
<b>3.1. Overview of Firmware Library of Standard Peripherals.....</b>	<b>46</b>
<b>3.2. ADC .....</b>	<b>46</b>
3.2.1. Descriptions of Peripheral registers.....	46
3.2.2. Descriptions of Peripheral functions .....	47
<b>3.3. CAU .....</b>	<b>74</b>
3.3.1. Descriptions of Peripheral registers.....	74
3.3.2. Descriptions of Peripheral functions .....	75
<b>3.4. CRC .....</b>	<b>102</b>
3.4.1. Descriptions of Peripheral registers.....	102
3.4.2. Descriptions of Peripheral functions .....	102
<b>3.5. CTC.....</b>	<b>107</b>
3.5.1. Descriptions of Peripheral registers.....	107
3.5.2. Descriptions of Peripheral functions .....	107
<b>3.6. DBG .....</b>	<b>120</b>
3.6.1. Descriptions of Peripheral registers.....	120
3.6.2. Descriptions of Peripheral functions .....	120
<b>3.7. DCI.....</b>	<b>125</b>
3.7.1. Descriptions of Peripheral registers.....	125
3.7.2. Descriptions of Peripheral functions .....	126

<b>3.8. DMA</b>	<b>139</b>
3.8.1. Descriptions of Peripheral registers	139
3.8.2. Descriptions of Peripheral functions	140
<b>3.9. EFUSE</b>	<b>171</b>
3.9.1. Descriptions of Peripheral registers	172
3.9.2. Descriptions of Peripheral functions	172
<b>3.10. EXTI</b>	<b>186</b>
3.10.1. Descriptions of Peripheral registers	186
3.10.2. Descriptions of Peripheral functions	186
<b>3.11. FMC</b>	<b>197</b>
3.11.1. Descriptions of Peripheral registers	197
3.11.2. Descriptions of Peripheral functions	198
<b>3.12. FWDGT</b>	<b>224</b>
3.12.1. Descriptions of Peripheral registers	224
3.12.2. Descriptions of Peripheral functions	224
<b>3.13. GPIO</b>	<b>229</b>
3.13.1. Descriptions of Peripheral registers	230
3.13.2. Descriptions of Peripheral functions	230
<b>3.14. HAU</b>	<b>243</b>
3.14.1. Descriptions of Peripheral registers	243
3.14.2. Descriptions of Peripheral functions	243
<b>3.15. HPDF</b>	<b>262</b>
3.15.1. Descriptions of Peripheral registers	262
3.15.2. Descriptions of Peripheral functions	262
<b>3.16. I2C</b>	<b>318</b>
3.16.1. Descriptions of Peripheral registers	319
3.16.2. Descriptions of Peripheral functions	319
<b>3.17. ICACHE</b>	<b>358</b>
3.17.1. Descriptions of Peripheral registers	358
3.17.2. Descriptions of Peripheral functions	358
<b>3.18. MISC</b>	<b>369</b>
3.18.1. Descriptions of Peripheral registers	369
3.18.2. Descriptions of Peripheral functions	370
<b>3.19. PKCAU</b>	<b>377</b>
3.19.1. Descriptions of Peripheral registers	377
3.19.2. Descriptions of Peripheral functions	378
<b>3.20. PMU</b>	<b>406</b>
3.20.1. Descriptions of Peripheral registers	406
3.20.2. Descriptions of Peripheral functions	407

<b>3.21.</b>	<b>QSPI .....</b>	<b>426</b>
3.21.1.	Descriptions of Peripheral registers .....	426
3.21.2.	Descriptions of Peripheral functions .....	426
<b>3.22.</b>	<b>RCU .....</b>	<b>441</b>
3.22.1.	Descriptions of Peripheral registers .....	441
3.22.2.	Descriptions of Peripheral functions .....	442
<b>3.23.</b>	<b>RTC .....</b>	<b>493</b>
3.23.1.	Descriptions of Peripheral registers .....	493
3.23.2.	Descriptions of Peripheral functions .....	494
<b>3.24.</b>	<b>SDIO .....</b>	<b>531</b>
3.24.1.	Descriptions of Peripheral registers .....	531
3.24.2.	Descriptions of Peripheral functions .....	532
<b>3.25.</b>	<b>SPI .....</b>	<b>562</b>
3.25.1.	Descriptions of Peripheral registers .....	562
3.25.2.	Descriptions of Peripheral functions .....	563
<b>3.26.</b>	<b>SQPI .....</b>	<b>589</b>
3.26.1.	Descriptions of Peripheral registers .....	589
3.26.2.	Descriptions of Peripheral functions .....	590
<b>3.27.</b>	<b>SYSCFG .....</b>	<b>596</b>
3.27.1.	Descriptions of Peripheral registers .....	596
3.27.2.	Descriptions of Peripheral functions .....	596
<b>3.28.</b>	<b>TIMER .....</b>	<b>610</b>
3.28.1.	Descriptions of Peripheral registers .....	610
3.28.2.	Descriptions of Peripheral functions .....	611
<b>3.29.</b>	<b>TRNG .....</b>	<b>666</b>
3.29.1.	Descriptions of Peripheral registers .....	666
3.29.2.	Descriptions of Peripheral functions .....	667
<b>3.30.</b>	<b>TSI .....</b>	<b>673</b>
3.30.1.	Descriptions of Peripheral registers .....	673
3.30.2.	Descriptions of Peripheral functions .....	673
<b>3.31.</b>	<b>TZPCU .....</b>	<b>693</b>
3.31.1.	Descriptions of Peripheral registers .....	693
3.31.2.	Descriptions of Peripheral functions .....	694
<b>3.32.</b>	<b>USART .....</b>	<b>717</b>
3.32.1.	Descriptions of Peripheral registers .....	717
3.32.2.	Descriptions of Peripheral functions .....	717
<b>3.33.</b>	<b>WWDGT .....</b>	<b>764</b>
3.33.1.	Descriptions of Peripheral registers .....	764
3.33.2.	Descriptions of Peripheral functions .....	764



4. Revision history ..... 769

# List of Figures

Figure 2-1. File structure of firmware library of GD32W51x_F5HC .....	37
Figure 2-2. Select peripheral example files.....	39
Figure 2-3. Copy the peripheral example files .....	39
Figure 2-4. Open the project file.....	40
Figure 2-5. Configure project files .....	41
Figure 2-6. Compile-debug-download .....	41
Figure 2-7. Select peripheral example files.....	42
Figure 2-8. Copy the peripheral example files .....	43
Figure 2-9. Open the workspace .....	43
Figure 2-10. Configure project files .....	44
Figure 2-11. Compile-debug-download .....	44

# List of Tables

Table 1-1. Peripherals.....	34
Table 2-1. Function descriptions of Firmware Library .....	45
Table 3-1. Peripheral function format of Firmware Library .....	46
Table 3-2. ADC Registers .....	46
Table 3-3. ADC firmware function .....	47
Table 3-4. Function adc_deinit .....	48
Table 3-5. Function adc_clock_config.....	49
Table 3-6. Function adc_enable .....	49
Table 3-7. Function adc_disable .....	50
Table 3-8. Function adc_dma_mode_enable .....	50
Table 3-9. Function adc_dma_mode_disable .....	51
Table 3-10. Function adc_dma_request_after_last_enable.....	52
Table 3-11. Function adc_dma_request_after_last_disable.....	52
Table 3-12. Function adc_end_of_conversion_config.....	53
Table 3-13. Function adc_internal_channel_config .....	53
Table 3-14. Function adc_discontinuous_mode_config.....	54
Table 3-15. Function adc_special_function_config .....	55
Table 3-16. Function adc_data_alignment_config .....	56
Table 3-17. Function adc_channel_length_config .....	56
Table 3-18. Function adc_routine_channel_config .....	57
Table 3-19. Function adc_inserted_channel_config .....	58
Table 3-20. Function adc_inserted_channel_offset_config .....	59
Table 3-21. Function adc_external_trigger_config.....	59
Table 3-22. Function adc_external_trigger_source_config.....	60
Table 3-23. Function adc_software_trigger_enable .....	62
Table 3-24. Function adc_routine_data_read .....	63
Table 3-25. Function adc_inserted_data_read.....	63
Table 3-26. Function adc_inserted_data_read.....	64
Table 3-27. Function adc_latch_data_read .....	64
Table 3-28. Function adc_latch_data_souce_config.....	65
Table 3-29. Function adc_watchdog0_single_channel_enable .....	66
Table 3-30. Function adc_watchdog0_sequence_channel_enable .....	66
Table 3-31. Function adc_watchdog0_disable .....	67
Table 3-32. Function adc_watchdog0_threshold_config.....	67
Table 3-33. Function adc_oversample_mode_config .....	68
Table 3-34. Function adc_oversample_mode_enable.....	69
Table 3-35. Function adc_oversample_mode_disable.....	70
Table 3-36. Function adc_flag_get .....	70
Table 3-37. Function adc_flag_clear .....	71
Table 3-38. Function adc_interrupt_enable .....	72
Table 3-39. Function adc_interrupt_disable .....	72

Table 3-40. Function <code>adc_interrupt_flag_get</code> .....	73
Table 3-41. Function <code>adc_interrupt_flag_clear</code> .....	74
Table 3-42. CAU Registers .....	74
Table 3-43. CAU firmware function .....	75
Table 3-44. Structure <code>cau_key_parameter_struct</code> .....	76
Table 3-45. Structure <code>cau_iv_parameter_struct</code> .....	76
Table 3-46. Structure <code>cau_context_parameter_struct</code> .....	77
Table 3-47. Structure <code>cau_parameter_struct</code> .....	77
Table 3-48. Function <code>cau_deinit</code> .....	77
Table 3-49. Function <code>cau_struct_para_init</code> .....	78
Table 3-50. Function <code>cau_key_struct_para_init</code> .....	79
Table 3-51. Function <code>cau_iv_struct_para_init</code> .....	79
Table 3-52. Function <code>cau_context_struct_para_init</code> .....	80
Table 3-53. Function <code>cau_enable</code> .....	80
Table 3-54. Function <code>cau_disable</code> .....	81
Table 3-55. Function <code>cau_dma_enable</code> .....	81
Table 3-56. Function <code>cau_dma_disable</code> .....	82
Table 3-57. Function <code>cau_init</code> .....	82
Table 3-58. Function <code>cau_aes_keysize_config</code> .....	84
Table 3-59. Function <code>cau_key_init</code> .....	84
Table 3-60. Function <code>cau_iv_init</code> .....	85
Table 3-61. Function <code>cau_phase_config</code> .....	85
Table 3-62. Function <code>cau_fifo_flush</code> .....	86
Table 3-63. Function <code>cau_enable_state_get</code> .....	86
Table 3-64. Function <code>cau_data_write</code> .....	87
Table 3-65. Function <code>cau_data_read</code> .....	87
Table 3-66. Function <code>cau_context_save</code> .....	88
Table 3-67. Function <code>cau_context_restore</code> .....	89
Table 3-68. Function <code>cau_aes_ecb</code> .....	89
Table 3-69. Function <code>cau_aes_cbc</code> .....	90
Table 3-70. Function <code>cau_aes_ctr</code> .....	91
Table 3-71. Function <code>cau_aes_cfb</code> .....	92
Table 3-72. Function <code>cau_aes_ofb</code> .....	93
Table 3-73. Function <code>cau_aes_gcm</code> .....	94
Table 3-74. Function <code>cau_aes_ccm</code> .....	95
Table 3-75. Function <code>cau_tdes_ecb</code> .....	96
Table 3-76. Function <code>cau_tdes_cbc</code> .....	97
Table 3-77. Function <code>cau_des_ecb</code> .....	98
Table 3-78. Function <code>cau_des_cbc</code> .....	99
Table 3-79. Function <code>cau_interrupt_enable</code> .....	100
Table 3-80. Function <code>cau_interrupt_disable</code> .....	100
Table 3-81. Function <code>cau_interrupt_flag_get</code> .....	101
Table 3-82. Function <code>cau_flag_get</code> .....	101
Table 3-83. CRC Registers .....	102

Table 3-84. CRC firmware function .....	102
Table 3-85. Function crc_deinit .....	103
Table 3-86. Function crc_data_register_reset .....	103
Table 3-87. Function crc_data_register_read .....	104
Table 3-88. Function crc_free_data_register_read.....	104
Table 3-89. Function crc_free_data_register_write.....	105
Table 3-90. Function crc_single_data_calculate .....	105
Table 3-91. Function crc_block_data_calculate .....	106
Table 3-92. CTC Registers .....	107
Table 3-93. CTC firmware function.....	107
Table 3-94. Function ctc_deinit .....	108
Table 3-95. Function ctc_counter_enable .....	108
Table 3-96. Function ctc_counter_disable .....	109
Table 3-97. Function ctc_irc48m_trim_value_config .....	109
Table 3-98. Function ctc_software_refsource_pulse_generate .....	110
Table 3-99. Function ctc_hardware_trim_mode_config .....	110
Table 3-100. Function ctc_refsource_polarity_config .....	111
Table 3-101. Function ctc_refsource_signal_select.....	111
Table 3-102. Function ctc_refsource_prescaler_config .....	112
Table 3-103. Function ctc_clock_limit_value_config .....	113
Table 3-104. Function ctc_counter_reload_value_config.....	113
Table 3-105. Function ctc_counter_capture_value_read.....	114
Table 3-106. Function ctc_counter_direction_read.....	114
Table 3-107. Function ctc_counter_reload_value_read .....	115
Table 3-108. Function ctc_irc48m_trim_value_read .....	115
Table 3-109. Function ctc_flag_get .....	116
Table 3-110. Function ctc_flag_clear .....	116
Table 3-111. Function ctc_interrupt_enable .....	117
Table 3-112. Function ctc_interrupt_disable.....	118
Table 3-113. Function ctc_interrupt_flag_get .....	118
Table 3-114. Function ctc_interrupt_flag_clear .....	119
Table 3-115. DBG Registers .....	120
Table 3-116. DBG firmware function .....	120
Table 3-117. Enum dbg_periph_enum .....	120
Table 3-118. Function dbg_deinit .....	121
Table 3-119. Function dbg_id_get .....	121
Table 3-120. Function dbg_low_power_enable .....	122
Table 3-121. Function dbg_low_power_disable .....	123
Table 3-122. Function dbg_periph_enable .....	123
Table 3-123. Function dbg_periph_disable .....	124
Table 3-124. Function dbg_trace_pin_enable .....	124
Table 3-125. Function dbg_trace_pin_disable .....	125
Table 3-126. DCI Registers.....	125
Table 3-127. DCI firmware function .....	126

Table 3-128. Structure dci_parameter_struct .....	127
Table 3-129. Function dci_deinit .....	127
Table 3-130. Function dci_struct_para_init.....	128
Table 3-131. Function dci_init .....	128
Table 3-132. Function dci_enable .....	129
Table 3-133. Function dci_disable .....	129
Table 3-134. Function dci_capture_enable .....	130
Table 3-135. Function dci_capture_disable .....	130
Table 3-136. Function dci_jpeg_enable .....	131
Table 3-137. Function dci_jpeg_disable .....	131
Table 3-138. Function dci_crop_window_enable .....	132
Table 3-139. Function dci_crop_window_disable .....	132
Table 3-140. Function dci_crop_window_config.....	133
Table 3-141. Function dci_embedded_sync_enable .....	133
Table 3-142. Function dci_embedded_sync_disable .....	134
Table 3-143. Function dci_sync_codes_config .....	134
Table 3-144. Function dci_sync_codes_unmask_config.....	135
Table 3-145. Function dci_data_read .....	135
Table 3-146. Function dci_flag_get .....	136
Table 3-147. Function dci_interrupt_enable.....	137
Table 3-148. Function dci_interrupt_disable.....	137
Table 3-149. Function dci_interrupt_flag_get .....	138
Table 3-150. Function dci_interrupt_flag_clear .....	138
Table 3-151. DMA Registers.....	139
Table 3-152. DMA firmware function .....	140
Table 3-153. Enum dma_channel_enum.....	141
Table 3-154. Enum dma_subperipheral_enum .....	141
Table 3-155. Structure dma_multi_data_parameter_struct .....	142
Table 3-156. Structure dma_single_data_parameter_struct .....	142
Table 3-157. Function dma_deinit .....	143
Table 3-158. Function dma_single_data_para_struct_init.....	143
Table 3-159. Function dma_multi_data_para_struct_init .....	144
Table 3-160. Function dma_single_data_mode_init.....	144
Table 3-161. Function dma_multi_data_mode_init .....	145
Table 3-162. Function dma_periph_address_config .....	146
Table 3-163. Function dma_memory_address_config .....	147
Table 3-164. Function dma_transfer_number_config .....	148
Table 3-165. Function dma_transfer_number_get .....	148
Table 3-166. Function dma_priority_config .....	149
Table 3-167. Function dma_memory_burst_beats_config .....	150
Table 3-168. Function dma_periph_burst_beats_config .....	150
Table 3-169. Function dma_memory_width_config .....	151
Table 3-170. Function dma_periph_width_config .....	152
Table 3-171. Function dma_memory_address_generation_config .....	153

Table 3-172. Function dma_peripheral_address_generation_config .....	154
Table 3-173. Function dma_circulation_enable .....	154
Table 3-174. Function dma_circulation_disable .....	155
Table 3-175. Function dma_channel_enable .....	156
Table 3-176. Function dma_channel_disable .....	156
Table 3-177. Function dma_transfer_direction_config .....	157
Table 3-178. Function dma_switch_buffer_mode_config .....	158
Table 3-179. Function dma_using_memory_get .....	158
Table 3-180. Function dma_channel_subperipheral_select .....	159
Table 3-181. Function dma_flow_controller_config .....	160
Table 3-182. Function dma_switch_buffer_mode_enable .....	161
Table 3-183. Function dma_switch_buffer_mode_enable .....	161
Table 3-184. Function dma_switch_buffer_mode_disable .....	162
Table 3-185. Function dma_switch_buffer_mode_disable .....	162
Table 3-186. Function dma_switch_buffer_mode_disable .....	163
Table 3-187. Function dma_fifo_status_get .....	163
Table 3-188. Function dma_flag_get .....	164
Table 3-189. Function dma_flag_clear .....	165
Table 3-190. Function dma_interrupt_flag_get .....	166
Table 3-191. Function dma_interrupt_flag_clear .....	166
Table 3-192. Function dma_interrupt_enable .....	167
Table 3-193. Function dma_interrupt_disable .....	168
Table 3-194. Function dma_security_config .....	169
Table 3-195. Function dma_priv_config .....	170
Table 3-196. Function dma_illegal_access_flag_get .....	170
Table 3-197. Function dma_illegal_access_flag_clear .....	171
Table 3-198. EFUSE Registers .....	172
Table 3-199. EFUSE firmware function .....	172
Table 3-200. Enum efuse_flag_enum .....	173
Table 3-201. Enum efuse_clear_flag_enum .....	173
Table 3-202. Enum efuse_int_enum .....	173
Table 3-203. Enum efuse_int_flag_enum .....	174
Table 3-204. efuse_clear_int_flag_enum .....	174
Table 3-205. Enum efuse_tz_enum .....	174
Table 3-206. Function efuse_read .....	174
Table 3-207. Function efuse_write .....	175
Table 3-208. Function efuse_boot_config .....	175
Table 3-209. Function efuse_tz_control_config .....	176
Table 3-210. Function efuse_fp_config .....	177
Table 3-211. Function efuse_mcu_init_data_write .....	177
Table 3-212. Function efuse_aes_key_write .....	178
Table 3-213. Function efuse_rotpk_key_write .....	178
Table 3-214. Function efuse_dp_write .....	179
Table 3-215. Function efuse_iak_write .....	180

Table 3-216. Function efuse_user_data_write .....	180
Table 3-217. Function efuse_software_trustzone_enable .....	181
Table 3-218. Function efuse_software_trustzone_disable .....	181
Table 3-219. Function efuse_boot_address_get .....	182
Table 3-220. Function efuse_flag_get .....	182
Table 3-221. Function efuse_flag_clear .....	183
Table 3-222. Function efuse_interrupt_enable .....	184
Table 3-223. Function efuse_interrupt_disable .....	184
Table 3-224. Function efuse_interrupt_flag_get .....	185
Table 3-225. Function efuse_interrupt_flag_clear .....	185
Table 3-226. EXTI Registers .....	186
Table 3-227. EXTI firmware function .....	186
Table 3-228. Enum exti_line_enum .....	187
Table 3-229. Enum exti_mode_enum .....	188
Table 3-230. Enum exti_trig_type_enum .....	188
Table 3-231. Function exti_deinit .....	188
Table 3-232. Function exti_init .....	189
Table 3-233. Function exti_interrupt_enable .....	189
Table 3-234. Function exti_interrupt_disable .....	190
Table 3-235. Function exti_event_enable .....	190
Table 3-236. Function exti_event_disable .....	191
Table 3-237. Function exti_security_enable .....	191
Table 3-238. Function exti_security_disable .....	192
Table 3-239. Function exti_privilege_enable .....	192
Table 3-240. Function exti_privilege_disable .....	193
Table 3-241. Function exti_lock_enable .....	193
Table 3-242. Function exti_software_interrupt_enable .....	194
Table 3-243. Function exti_software_interrupt_disable .....	194
Table 3-244. Function exti_flag_get .....	195
Table 3-245. Function exti_flag_clear .....	195
Table 3-246. Function exti_interrupt_flag_get .....	196
Table 3-247. Function exti_interrupt_flag_clear .....	196
Table 3-248. FMC Registers .....	197
Table 3-249. FMC firmware function .....	198
Table 3-250. fmc_state_enum .....	199
Table 3-251. Function fmc_unlock .....	199
Table 3-252. Function fmc_lock .....	200
Table 3-253. Function fmc_page_erase .....	200
Table 3-254. Function fmc_mass_erase .....	201
Table 3-255. Function fmc_word_program .....	201
Table 3-256. Function fmc_continuous_program .....	202
Table 3-257. Function sram1_reset_enable .....	203
Table 3-258. Function sram1_reset_disable .....	203
Table 3-259. Function fmc_privilege_enable .....	204



Table 3-260. Function fmc_privilege_disable .....	204
Table 3-261. Function ob_unlock .....	205
Table 3-262. Function ob_lock .....	205
Table 3-263. Function ob_start .....	206
Table 3-264. Function ob_reload .....	206
Table 3-265. Function ob_security_protection_config .....	207
Table 3-266. Function ob_wdgt_config .....	207
Table 3-267. Function ob_boot_config .....	208
Table 3-268. Function ob_trustzone_enable .....	209
Table 3-269. Function ob_trustzone_disable .....	210
Table 3-270. Function ob_user_write .....	210
Table 3-271. Function ob_write_protection_config .....	211
Table 3-272. Function ob_secmark_config .....	212
Table 3-273. Function ob_dmp_access_enable .....	212
Table 3-274. Function ob_dmp_access_disable .....	213
Table 3-275. Function ob_dmp_config .....	213
Table 3-276. Function ob_dmp_enable .....	214
Table 3-277. Function ob_dmp_disable .....	215
Table 3-278. Function fmc_no_rtdec_config .....	215
Table 3-279. Function fmc_offset_region_config .....	216
Table 3-280. Function fmc_offset_value_config .....	216
Table 3-281. Function ob_write_protection_get .....	217
Table 3-282. Function ob_user_get .....	217
Table 3-283. Function ob_security_protection_flag_get .....	218
Table 3-284. Function ob_trustzone_state_get .....	219
Table 3-285. Function ob_exist_state_get .....	219
Table 3-286. Function fmc_flag_get .....	220
Table 3-287. Function fmc_flag_clear .....	220
Table 3-288. Function fmc_interrupt_enable .....	221
Table 3-289. Function fmc_interrupt_disable .....	221
Table 3-290. Function fmc_interrupt_flag_get .....	222
Table 3-291. Function fmc_interrupt_flag_clear .....	223
Table 3-292. Function fmc_ready_wait .....	223
Table 3-293. FWDGT Registers .....	224
Table 3-294. FWDGT firmware function .....	224
Table 3-295. Function fwdgt_write_enable .....	225
Table 3-296. Function fwdgt_write_disable .....	225
Table 3-297. Function fwdgt_enable .....	226
Table 3-298. Function fwdgt_prescaler_value_config .....	226
Table 3-299. Function fwdgt_reload_value_config .....	227
Table 3-300. Function fwdgt_window_value_config .....	227
Table 3-301. Function fwdgt_counter_reload .....	228
Table 3-302. Function fwdgt_config .....	228
Table 3-303. Function fwdgt_flag_get .....	229

Table 3-304. GPIO Registers .....	230
Table 3-305. GPIO firmware function .....	230
Table 3-306. Function gpio_deinit.....	231
Table 3-307. Function gpio_mode_set.....	231
Table 3-308. Function gpio_output_options_set .....	232
Table 3-309. Function gpio_bit_set.....	233
Table 3-310. Function gpio_bit_reset .....	234
Table 3-311. Function gpio_bit_write.....	234
Table 3-312. Function gpio_port_write .....	235
Table 3-313. Function gpio_input_bit_get.....	236
Table 3-314. Function gpio_input_port_get .....	236
Table 3-315. Function gpio_output_bit_get .....	237
Table 3-316. Function gpio_output_port_get.....	237
Table 3-317. Function gpio_af_set .....	238
Table 3-318. Function gpio_pin_lock.....	239
Table 3-319. Function gpio_bit_toggle .....	240
Table 3-320. Function gpio_port_toggle.....	240
Table 3-321. Function gpio_bit_set_sec_cfg .....	241
Table 3-322. Function gpio_bit_reset_sec_cfg.....	241
Table 3-323. Function gpio_sec_cfg_bit_get .....	242
Table 3-324. HAU Registers .....	243
Table 3-325. HAU firmware function .....	243
Table 3-326. Structure hau_init_parameter_struct.....	244
Table 3-327. Structure hau_digest_parameter_struct.....	244
Table 3-328. Structure hau_context_parameter_struct .....	245
Table 3-329. Function hau_deinit.....	245
Table 3-330. Function hau_init .....	245
Table 3-331. Function hau_init_struct_para_init .....	246
Table 3-332. Function hau_reset .....	247
Table 3-333. Function hau_last_word_validbits_num_config.....	247
Table 3-334. Function hau_data_write.....	248
Table 3-335. Function hau_infifo_words_num_get .....	248
Table 3-336. Function hau_digest_read .....	249
Table 3-337. Function hau_digest_calculation_enable.....	249
Table 3-338. Function hau_multiple_single_dma_config.....	250
Table 3-339. Function hau_dma_enable.....	250
Table 3-340. Function hau_dma_disable .....	251
Table 3-341. Function hau_context_struct_para_init.....	251
Table 3-342. Function hau_context_save.....	252
Table 3-343. Function hau_context_restore .....	252
Table 3-344. Function hau_hash_sha_1 .....	253
Table 3-345. Function hau_hmac_sha_1 .....	253
Table 3-346. Function hau_hash_sha_224.....	254
Table 3-347. Function hau_hmac_sha_224 .....	255

Table 3-348. Function hau_hash_sha_256 .....	255
Table 3-349. Function hau_hmac_sha_256 .....	256
Table 3-350. Function hau_hash_md5 .....	257
Table 3-351. Function hau_hmac_md5 .....	257
Table 3-352. Function hau_flag_get .....	258
Table 3-353. Function hau_flag_clear .....	258
Table 3-354. Function hau_interrupt_enable .....	259
Table 3-355. Function hau_interrupt_disable .....	260
Table 3-356. Function hau_interrupt_flag_get .....	260
Table 3-357. Function hau_interrupt_flag_clear .....	261
Table 3-358. HPDF Registers .....	262
Table 3-359. HPDF firmware function .....	262
Table 3-360. Enum hpdf_channel_enum .....	265
Table 3-361. Enum hpdf_filter_enum .....	265
Table 3-362. Enum hpdf_flag_enum .....	265
Table 3-363. Enum hpdf_interrput_flag_enum .....	266
Table 3-364. Enum hpdf_interrput_enum .....	267
Table 3-365. Structure hpdf_channel_parameter_struct .....	267
Table 3-366. Structure hpdf_filter_parameter_struct .....	267
Table 3-367. Structure hpdf_rc_parameter_struct .....	268
Table 3-368. Structure hpdf_ic_parameter_struct .....	268
Table 3-369. Function hpdf_deinit .....	268
Table 3-370. Function hpdf_channel_struct_para_init .....	269
Table 3-371. Function hpdf_filter_struct_para_init .....	269
Table 3-372. Function hpdf_rc_struct_para_init .....	270
Table 3-373. Function hpdf_ic_struct_para_init .....	270
Table 3-374. Function hpdf_enable .....	271
Table 3-375. Function hpdf_disable .....	271
Table 3-376. Function hpdf_channel_init .....	272
Table 3-377. Function hpdf_filter_init .....	273
Table 3-378. Function hpdf_rc_init .....	274
Table 3-379. Function hpdf_ic_init .....	275
Table 3-380. Function hpdf_clock_output_config .....	275
Table 3-381. Function hpdf_clock_output_source_config .....	276
Table 3-382. Function hpdf_clock_output_duty_mode_disable .....	277
Table 3-383. Function hpdf_clock_output_duty_mode_enable .....	277
Table 3-384. Function hpdf_clock_output_divider_config .....	277
Table 3-385. Function hpdf_channel_enable .....	278
Table 3-386. Function hpdf_channel_disable .....	278
Table 3-387. Function hpdf_spi_clock_source_config .....	279
Table 3-388. Function hpdf_serial_interface_type_config .....	280
Table 3-389. Function hpdf_malfunction_monitor_disable .....	280
Table 3-390. Function hpdf_malfunction_monitor_enable .....	281
Table 3-391. Function hpdf_clock_loss_disable .....	281

Table 3-392. Function <code>hpdf_clock_loss_enable</code> .....	282
Table 3-393. Function <code>hpdf_channel_pin_redirection_disable</code> .....	282
Table 3-394. Function <code>hpdf_channel_pin_redirection_enable</code> .....	283
Table 3-395. Function <code>hpdf_channel_multiplexer_config</code> .....	283
Table 3-396. Function <code>hpdf_data_pack_mode_config</code> .....	284
Table 3-397. Function <code>hpdf_data_right_bit_shift_config</code> .....	285
Table 3-398. Function <code>hpdf_calibration_offset_config</code> .....	285
Table 3-399. Function <code>hpdf_malfunction_break_signal_config</code> .....	286
Table 3-400. Function <code>hpdf_malfunction_counter_config</code> .....	286
Table 3-401. Function <code>hpdf_write_parallel_data_standard_mode</code> .....	287
Table 3-402. Function <code>hpdf_write_parallel_data_interleaved_mode</code> .....	288
Table 3-403. Function <code>hpdf_write_parallel_data_dual_mode</code> .....	288
Table 3-404. Function <code>hpdf_pulse_skip_update</code> .....	289
Table 3-405. Function <code>hpdf_pulse_skip_read</code> .....	289
Table 3-406. Function <code>hpdf_filter_enable</code> .....	290
Table 3-407. Function <code>hpdf_filter_disable</code> .....	290
Table 3-408. Function <code>hpdf_filter_config</code> .....	291
Table 3-409. Function <code>hpdf_integrator_oversample</code> .....	292
Table 3-410. Function <code>hpdf_threshold_monitor_filter_config</code> .....	292
Table 3-411. Function <code>hpdf_threshold_monitor_filter_read_data</code> .....	293
Table 3-412. Function <code>hpdf_threshold_monitor_fast_mode_disable</code> .....	293
Table 3-413. Function <code>hpdf_threshold_monitor_fast_mode_enable</code> .....	294
Table 3-414. Function <code>hpdf_threshold_monitor_channel</code> .....	294
Table 3-415. Function <code>hpdf_threshold_monitor_high_threshold</code> .....	295
Table 3-416. Function <code>hpdf_threshold_monitor_low_threshold</code> .....	296
Table 3-417. Function <code>hpdf_high_threshold_break_signal</code> .....	296
Table 3-418. Function <code>hpdf_low_threshold_break_signal</code> .....	297
Table 3-419. Function <code>hpdf_extremes_monitor_channel</code> .....	298
Table 3-420. Function <code>hpdf_extremes_monitor_maximum_get</code> .....	298
Table 3-421. Function <code>hpdf_extremes_monitor_minimum_get</code> .....	299
Table 3-422. Function <code>hpdf_rc_continuous_disable</code> .....	299
Table 3-423. Function <code>hpdf_rc_continuous_enable</code> .....	300
Table 3-424. Function <code>hpdf_rc_start_by_software</code> .....	300
Table 3-425. Function <code>hpdf_rc_syn_disable</code> .....	301
Table 3-426. Function <code>hpdf_rc_syn_enable</code> .....	301
Table 3-427. Function <code>hpdf_rc_dma_disable</code> .....	302
Table 3-428. Function <code>hpdf_rc_dma_enable</code> .....	302
Table 3-429. Function <code>hpdf_rc_channel_config</code> .....	303
Table 3-430. Function <code>hpdf_rc_fast_mode_disable</code> .....	304
Table 3-431. Function <code>hpdf_rc_fast_mode_enable</code> .....	304
Table 3-432. Function <code>hpdf_rc_data_get</code> .....	305
Table 3-433. Function <code>hpdf_rc_channel_get</code> .....	305
Table 3-434. Function <code>hpdf_ic_start_by_software</code> .....	306
Table 3-435. Function <code>hpdf_ic_syn_disable</code> .....	306

Table 3-436. Function hpdf_ic_syn_enable .....	307
Table 3-437. Function hpdf_ic_dma_disable .....	307
Table 3-438. Function hpdf_ic_dma_enable .....	308
Table 3-439. Function hpdf_ic_scan_mode_disable .....	308
Table 3-440. Function hpdf_ic_scan_mode_enable .....	309
Table 3-441. Function hpdf_ic_trigger_signal_disable .....	309
Table 3-442. Function hpdf_ic_trigger_signal_config .....	310
Table 3-443. Function hpdf_ic_channel_config .....	310
Table 3-444. Function hpdf_ic_data_get .....	311
Table 3-445. Function hpdf_ic_channel_get .....	312
Table 3-446. Function hpdf_flag_get .....	312
Table 3-447. Function hpdf_flag_clear .....	313
Table 3-448. Function hpdf_interrupt_enable .....	315
Table 3-449. Function hpdf_interrupt_disable .....	316
Table 3-450. Function hpdf_interrupt_flag_get .....	317
Table 3-451. Function hpdf_interrupt_flag_clear .....	318
Table 3-452. I2C Registers .....	319
Table 3-453. I2C firmware function .....	319
Table 3-454. i2c_interrupt_flag_enum .....	321
Table 3-455. Function i2c_deinit .....	321
Table 3-456. Function i2c_timing_config .....	322
Table 3-457. Function i2c_digital_noise_filter_config .....	323
Table 3-458. Function i2c_analog_noise_filter_enable .....	324
Table 3-459. Function i2c_analog_noise_filter_disable .....	324
Table 3-460. Function i2c_master_clock_config .....	325
Table 3-461. Function i2c_master_addressing .....	325
Table 3-462. Function i2c_address10_header_enable .....	326
Table 3-463. Function i2c_address10_header_disable .....	326
Table 3-464. Function i2c_address10_enable .....	327
Table 3-465. Function i2c_address10_disable .....	327
Table 3-466. Function i2c_automatic_end_enable .....	328
Table 3-467. Function i2c_automatic_end_disable .....	328
Table 3-468. Function i2c_slave_response_to_gcall_enable .....	329
Table 3-469. Function i2c_slave_response_to_gcall_disable .....	329
Table 3-470. Function i2c_stretch_scl_low_enable .....	330
Table 3-471. Function i2c_stretch_scl_low_disable .....	330
Table 3-472. Function i2c_address_config .....	331
Table 3-473. Function i2c_address_bit_compare_config .....	332
Table 3-474. Function i2c_address_disable .....	333
Table 3-475. Function i2c_second_address_config .....	333
Table 3-476. Function i2c_second_address_disable .....	334
Table 3-477. Function i2c_received_address_get .....	335
Table 3-478. Function i2c_slave_byte_control_enable .....	335
Table 3-479. Function i2c_slave_byte_control_disable .....	336

Table 3-480. Function i2c_nack_enable .....	336
Table 3-481. Function i2c_nack_disable .....	337
Table 3-482. Function i2c_wakeup_from_deepsleep_enable .....	337
Table 3-483. Function i2c_wakeup_from_deepsleep_disable .....	338
Table 3-484. Function i2c_enable.....	338
Table 3-485. Function i2c_disable.....	339
Table 3-486. Function i2c_start_on_bus .....	339
Table 3-487. Function i2c_stop_on_bus.....	340
Table 3-488. Function i2c_data_transmit .....	340
Table 3-489. Function i2c_data_receive .....	341
Table 3-490. Function i2c_reload_enable.....	341
Table 3-491. Function i2c_reload_disable.....	342
Table 3-492. Function i2c_transfer_byte_number_config .....	342
Table 3-493. Function i2c_dma_enable .....	343
Table 3-494. Function i2c_dma_disable .....	343
Table 3-495. Function i2c_pec_transfer .....	344
Table 3-496. Function i2c_pec_enable .....	344
Table 3-497. Function i2c_pec_disable .....	345
Table 3-498. Function i2c_pec_value_get .....	345
Table 3-499. Function i2c_smbus_alert_enable .....	346
Table 3-500. Function i2c_smbus_alert_disable .....	347
Table 3-501. Function i2c_smbus_default_addr_enable .....	347
Table 3-502. Function i2c_smbus_default_addr_disable .....	348
Table 3-503. Function i2c_smbus_host_addr_enable .....	348
Table 3-504. Function i2c_smbus_host_addr_disable .....	349
Table 3-505. Function i2c_extented_clock_timeout_enable .....	349
Table 3-506. Function i2c_extented_clock_timeout_disable .....	350
Table 3-507. Function i2c_clock_timeout_enable .....	350
Table 3-508. Function i2c_clock_timeout_disable .....	351
Table 3-509. Function i2c_bus_timeout_b_config .....	351
Table 3-510. Function i2c_bus_timeout_a_config.....	352
Table 3-511. Function i2c_idle_clock_timeout_config.....	352
Table 3-512. Function i2c_flag_get .....	353
Table 3-513. Function i2c_flag_clear .....	354
Table 3-514. Function i2c_interrupt_enable.....	355
Table 3-515. Function i2c_interrupt_disable.....	355
Table 3-516. Function i2c_interrupt_flag_get .....	356
Table 3-517. Function i2c_interrupt_flag_clear .....	357
Table 3-518. ICACHE Registers .....	358
Table 3-519. ICACHE firmware function .....	359
Table 3-520. Structure icache_remap_struct .....	359
Table 3-521. Function icache_enable .....	359
Table 3-522. Function icache_disable .....	360
Table 3-523. Function icache_monitor_enable .....	360

Table 3-524. Function <code>icache_monitor_disable</code> .....	361
Table 3-525. Function <code>icache_monitor_reset</code> .....	362
Table 3-526. Function <code>icache_way_configure</code> .....	362
Table 3-527. Function <code>icache_burst_type_select</code> .....	363
Table 3-528. Function <code>icache_invalidation</code> .....	363
Table 3-529. Function <code>icache_hitvalue_get</code> .....	364
Table 3-530. Function <code>icache_missvalue_get</code> .....	364
Table 3-531. Function <code>icache_remap_enable</code> .....	365
Table 3-532. Function <code>icache_remap_disable</code> .....	365
Table 3-533. Function <code>icache_flag_get</code> .....	366
Table 3-534. Function <code>icache_flag_clear</code> .....	366
Table 3-535. Function <code>icache_interrupt_enable</code> .....	367
Table 3-536. Function <code>icache_interrupt_disable</code> .....	368
Table 3-537. Function <code>icache_interrupt_flag_get</code> .....	368
Table 3-538. Function <code>icache_interrupt_flag_clear</code> .....	369
Table 3-539. NVIC Registers .....	369
Table 3-540. SysTick Registers .....	370
Table 3-541. MISC firmware function .....	370
Table 3-542. <code>IRQn_Type ()</code> .....	370
Table 3-543. Function <code>nvic_priority_group_set</code> .....	372
Table 3-544. Function <code>nvic_irq_enable</code> .....	373
Table 3-545. Function <code>nvic_irq_disable</code> .....	374
Table 3-546. Function <code>nvic_system_reset</code> .....	374
Table 3-547. Function <code>nvic_vector_table_set</code> .....	375
Table 3-548. Function <code>system_lowpower_set</code> .....	375
Table 3-549. Function <code>system_lowpower_reset</code> .....	376
Table 3-550. Function <code>systick_clksource_set</code> .....	377
Table 3-551. PKCAU Registers .....	377
Table 3-552. PKCAU firmware function .....	378
Table 3-553. Structure <code>pkcau_mont_parameter_struct</code> .....	379
Table 3-554. Structure <code>pkcau_mod_parameter_struct</code> .....	379
Table 3-555. Structure <code>pkcau_mod_exp_parameter_struct</code> .....	379
Table 3-556. Structure <code>pkcau_mod_inver_parameter_struct</code> .....	379
Table 3-557. Structure <code>pkcau_mod_reduc_parameter_struct</code> .....	379
Table 3-558. Structure <code>pkcau_arithmetic_parameter_struct</code> .....	380
Table 3-559. Structure <code>pkcau_crt_parameter_struct</code> .....	380
Table 3-560. Structure <code>pkcau_ec_group_parameter_struct</code> .....	380
Table 3-561. Structure <code>pkcau_point_parameter_struct</code> .....	381
Table 3-562. Structure <code>pkcau_signature_parameter_struct</code> .....	381
Table 3-563. Structure <code>pkcau_hash_parameter_struct</code> .....	381
Table 3-564. Function <code>pkcau_deinit</code> .....	381
Table 3-565. Function <code>pkcau_mont_struct_para_init</code> .....	382
Table 3-566. Function <code>pkcau_mod_struct_para_init</code> .....	382
Table 3-567. Function <code>pkcau_mod_exp_struct_para_init</code> .....	383

Table 3-568. Function pkcau_mod_inver_struct_para_init .....	383
Table 3-569. Function pkcau_arithmetic_struct_para_init .....	384
Table 3-570. Function pkcau_crt_struct_para_init.....	384
Table 3-571. Function pkcau_ec_group_struct_para_init .....	385
Table 3-572. Function pkcau_point_struct_para_init.....	385
Table 3-573. Function pkcau_signature_struct_para_init .....	386
Table 3-574. Function pkcau_hash_struct_para_init .....	387
Table 3-575. Function pkcau_mod_reduc_struct_para_init.....	387
Table 3-576. Function pkcau_enable .....	388
Table 3-577. Function pkcau_disable .....	388
Table 3-578. Function pkcau_start.....	389
Table 3-579. Function pkcau_mode_set.....	389
Table 3-580. Function pkcau_mont_param_operation .....	390
Table 3-581. Function pkcau_mod_operation .....	391
Table 3-582. Function pkcau_mod_exp_operation .....	392
Table 3-583. Function pkcau_mod_inver_operation.....	393
Table 3-584. Function pkcau_mod_reduc_operation.....	394
Table 3-585. Function pkcau_arithmetic_operation.....	395
Table 3-586. Function pkcau_crt_exp_operation .....	396
Table 3-587. Function pkcau_point_check_operation .....	397
Table 3-588. Function pkcau_point_mul_operation.....	398
Table 3-589. Function pkcau_ecdsa_sign_operation .....	399
Table 3-590. Function pkcau_ecdsa_verification_operation .....	400
Table 3-591. Function pkcau_memread.....	402
Table 3-592. Function pkcau_flag_get.....	403
Table 3-593. Function pkcau_flag_clear.....	403
Table 3-594. Function pkcau_interrupt_enable .....	404
Table 3-595. Function pkcau_interrupt_disable .....	405
Table 3-596. Function pkcau_interrupt_flag_get.....	405
Table 3-597. Function pkcau_interrupt_flag_clear.....	406
Table 3-598. PMU Registers .....	407
Table 3-599. PMU firmware function .....	407
Table 3-600. Function pmu_deinit.....	408
Table 3-601. Function pmu_lvd_select .....	408
Table 3-602. Function pmu_deepsleep_voltage_select.....	409
Table 3-603. Function pmu_lvd_disable.....	409
Table 3-604. Function pmu_vlvd_enable.....	410
Table 3-605. Function pmu_vlvd_disable.....	410
Table 3-606. Function pmu_ldo_output_select .....	411
Table 3-607. Function pmu_to_sleepmode .....	411
Table 3-608. Function pmu_to_deepsleepmode.....	412
Table 3-609. Function pmu_to_standbymode .....	413
Table 3-610. Function pmu_wakeup_pin_enable .....	413
Table 3-611. Function pmu_wakeup_pin_disable .....	414



Table 3-612. Function pmu_wakeup_pin_edge_select .....	415
Table 3-613. Function pmu_backup_write_enable .....	416
Table 3-614. Function pmu_backup_write_disable .....	416
Table 3-615. Function pmu_wifi_power_enable .....	417
Table 3-616. Function pmu_wifi_power_disable .....	417
Table 3-617. Function pmu_wifi_low_power_control .....	418
Table 3-618. Function pmu_sram_low_power_control .....	418
Table 3-619. Function pmu_rf_force_enable .....	419
Table 3-620. Function pmu_rf_force_disable .....	419
Table 3-621. Function pmu_rf_sequence_config .....	420
Table 3-622. Function pmu_security_enable .....	420
Table 3-623. Function pmu_security_disable .....	421
Table 3-624. Function pmu_privilege_enable .....	422
Table 3-625. Function pmu_privilege_disable .....	423
Table 3-626. Function pmu_flag_clear .....	423
Table 3-627. Function pmu_flag_get .....	424
Table 3-628. QSPI registers .....	426
Table 3-629. QSPI firmware function .....	426
Table 3-630. qspi_init_struct .....	427
Table 3-631. qspi_command_struct .....	427
Table 3-632. qspi_autopolling_struct .....	428
Table 3-633. Function qspi_deinit .....	429
Table 3-634. Function qspi_init_struct_para_init .....	429
Table 3-635. Function qspi_init .....	430
Table 3-636. Function quad_spi_enable .....	430
Table 3-637. Function qspi_disable .....	431
Table 3-638. Function qspi_dma_enable .....	431
Table 3-639. Function qspi_dma_disable .....	432
Table 3-640. Function qspi_data_length_config .....	432
Table 3-641. Function qspi_command .....	433
Table 3-642. Function qspi_transmit .....	434
Table 3-643. Function qspi_receive .....	434
Table 3-644. Function qspi_command .....	435
Table 3-645. Function qspi_memorymapped .....	436
Table 3-646. Function qspi_abort .....	437
Table 3-647. Function qspi_interrupt_enable .....	438
Table 3-648. Function qspi_interrupt_disable .....	438
Table 3-649. Function qspi_flag_get .....	439
Table 3-650. Function qspi_flag_clear .....	439
Table 3-651. Function qspi_command .....	440
Table 3-652. RCU Registers .....	441
Table 3-653. RCU firmware function .....	442
Table 3-654. Enum rcu_periph_enum .....	444
Table 3-655. Enum rcu_periph_sleep_enum .....	446

Table 3-656. Enum rcu_periph_reset_enum .....	447
Table 3-657. Enum rcu_flag_enum.....	448
Table 3-658. Enum rcu_int_flag_enum .....	449
Table 3-659. Enum rcu_int_flag_clear_enum.....	449
Table 3-660. Enum rcu_int_enum.....	450
Table 3-661. Enum rcu_osci_type_enum .....	450
Table 3-662. Enum rcu_clock_freq_enum .....	450
Table 3-663. Enum rcu_sec_enum .....	450
Table 3-664. Enum rcu_sec_flag_enum.....	451
Table 3-665. Enum rcu_unit_enum .....	453
Table 3-666. Function rcu_deinit.....	454
Table 3-667. Function rcu_periph_clock_enable.....	454
Table 3-668. Function rcu_periph_clock_disable.....	454
Table 3-669. Function rcu_periph_clock_sleep_enable .....	455
Table 3-670. Function rcu_periph_clock_sleep_disable .....	455
Table 3-671. Function rcu_periph_reset_enable .....	456
Table 3-672. Function rcu_periph_reset_disable .....	456
Table 3-673. Function rcu_bkp_reset_enable .....	457
Table 3-674. Function rcu_bkp_reset_disable .....	457
Table 3-675. Function rcu_hxtal_plli2s_enable .....	458
Table 3-676. Function rcu_hxtal_plli2s_disable .....	458
Table 3-677. Function rcu_hxtal_pllp_enable .....	459
Table 3-678. Function rcu_hxtal_pllp_disable .....	459
Table 3-679. Function rcu_control_unit_powerup .....	460
Table 3-680. Function rcu_control_unit_powerdown .....	460
Table 3-681. Function rcu_rfppl_cal_enable .....	461
Table 3-682. Function rcu_rfppl_cal_disable .....	461
Table 3-683. Function rcu_system_clock_source_config .....	462
Table 3-684. Function rcu_system_clock_source_get .....	463
Table 3-685. Function rcu_ahb_clock_config .....	463
Table 3-686. Function rcu_apb1_clock_config .....	464
Table 3-687. Function rcu_apb2_clock_config .....	464
Table 3-688. Function rcu_ckout0_config .....	465
Table 3-689. Function rcu_ckout1_config .....	466
Table 3-690. Function rcu_pll_config .....	466
Table 3-691. Function rcu_plli2s_config .....	467
Table 3-692. Function rcu_pllpresel_config.....	468
Table 3-693. Function rcu_plldig_div_sys_config.....	469
Table 3-694. Function rcu_rtc_clock_config.....	469
Table 3-695. Function rcu_rtc_div_config.....	470
Table 3-696. Function rcu_i2s_clock_config .....	470
Table 3-697. Function rcu_pll_div_i2s_config .....	471
Table 3-698. Function rcu_hpdpf_clock_config .....	471
Table 3-699. Function rcu_hpdpf_audio_clock_config.....	472

Table 3-700. Function rcu_sdio_clock_config .....	473
Table 3-701. Function rcu_sdio_div_config .....	473
Table 3-702. Function rcu_usbfs_clock_config .....	474
Table 3-703. Function rcu_usbfs_div_config .....	474
Table 3-704. Function rcu_i2c0_clock_config .....	475
Table 3-705. Function rcu_usart0_clock_config .....	475
Table 3-706. Function rcu_usart2_clock_config .....	476
Table 3-707. Function rcu_irc16m_div_config .....	477
Table 3-708. Function rcu_sdio_div_config .....	477
Table 3-709. Function rcu_lxtal_drive_capability_config .....	478
Table 3-710. Function rcu_osci_stab_wait .....	479
Table 3-711. Function rcu_osci_on .....	479
Table 3-712. Function rcu_osci_off .....	480
Table 3-713. Function rcu_osci_bypass_mode_enable .....	480
Table 3-714. Function rcu_osci_bypass_mode_disable .....	481
Table 3-715. Function rcu_hxtal_clock_monitor_enable .....	481
Table 3-716. Function rcu_hxtal_clock_monitor_disable .....	482
Table 3-717. Function rcu_hxtal_clock_monitor_enable .....	482
Table 3-718. Function rcu_hxtal_clock_monitor_disable .....	482
Table 3-719. Function rcu_IRC16M_adjust_value_set .....	483
Table 3-720. Function rcu_spread_spectrum_config .....	483
Table 3-721. Function rcu_spread_spectrum_enable .....	484
Table 3-722. Function rcu_spread_spectrum_disable .....	485
Table 3-723. Function rcu_voltage_key_unlock .....	485
Table 3-724. Function rcu_deepsleep_voltage_set .....	486
Table 3-725. Function rcu_clock_freq_get .....	486
Table 3-726. Function rcu_security_enable .....	487
Table 3-727. Function rcu_security_disable .....	487
Table 3-728. Function rcu_privilege_enable .....	488
Table 3-729. Function rcu_privilege_disable .....	488
Table 3-730. Function rcu_flag_get .....	489
Table 3-731. Function rcu_all_reset_flag_clear .....	489
Table 3-732. Function rcu_interrupt_flag_get .....	490
Table 3-733. Function rcu_interrupt_flag_clear .....	490
Table 3-734. Function rcu_security_flag_get .....	491
Table 3-735. Function rcu_interrupt_enable .....	492
Table 3-736. Function rcu_interrupt_disable .....	493
Table 3-737. RTC Registers .....	493
Table 3-738. RTC firmware function .....	494
Table 3-739. Structure rtc_parameter_struct .....	496
Table 3-740. Structure rtc_alarm_struct .....	496
Table 3-741. Structure rtc_timestamp_struct .....	496
Table 3-742. Structure rtc_tamper_struct .....	496
Table 3-743. Function rtc_deinit .....	497

Table 3-744. Function rtc_init .....	497
Table 3-745. Function rtc_init_mode_enter .....	498
Table 3-746. Function rtc_init_mode_exit .....	499
Table 3-747. Function rtc_register_sync_wait .....	499
Table 3-748. Function rtc_current_time_get .....	500
Table 3-749. Function rtc_subsecond_get .....	500
Table 3-750. Function rtc_alarm_config .....	501
Table 3-751. Function rtc_alarm_subsecond_config .....	501
Table 3-752. Function rtc_alarm_get .....	503
Table 3-753. Function rtc_alarm_subsecond_get .....	503
Table 3-754. Function rtc_alarm_enable .....	504
Table 3-755. Function rtc_alarm_disable .....	504
Table 3-756. Function rtc_timestamp_enable .....	505
Table 3-757. Function rtc_timestamp_disable .....	505
Table 3-758. Function rtc_timestamp_get .....	506
Table 3-759. Function rtc_timestamp_subsecond_get .....	507
Table 3-760. Function rtc_tamper_enable .....	507
Table 3-761. Function rtc_tamper_disable .....	508
Table 3-762. Function rtc_software_bkp_reset .....	508
Table 3-763. Function rtc_tamper_without_bkp_seset .....	509
Table 3-764. Function rtc_output_pin_select .....	509
Table 3-765. Function rtc_alter_output_config .....	510
Table 3-766. Function rtc_calibration_config .....	511
Table 3-767. Function rtc_hour_adjust .....	511
Table 3-768. Function rtc_second_adjust .....	512
Table 3-769. Function rtc_bypass_shadow_enable .....	512
Table 3-770. Function rtc_bypass_shadow_disable .....	513
Table 3-771. Function rtc_refclock_detection_enable .....	513
Table 3-772. Function rtc_refclock_detection_disable .....	514
Table 3-773. Function rtc_wakeup_enable .....	514
Table 3-774. Function rtc_wakeup_disable .....	515
Table 3-775. Function rtc_wakeup_clock_set .....	515
Table 3-776. Function rtc_wakeup_timer_set .....	516
Table 3-777. Function rtc_wakeup_timer_get .....	516
Table 3-778. rtc_smooth_calibration_config .....	517
Table 3-779. rtc_coarse_calibration_enable .....	518
Table 3-780. rtc_coarse_calibration_disable .....	518
Table 3-781. rtc_coarse_calibration_config .....	519
Table 3-782. rtc_pri_pro_enable .....	520
Table 3-783. rtc_pri_pro_enable .....	521
Table 3-784. rtc_sec_pro_enable .....	522
Table 3-785. rtc_sec_pro_disable .....	522
Table 3-786. rtc_bkp_zonea_sec_pro_set .....	523
Table 3-787. rtc_bkp_zoneb_sec_pro_set .....	524

Table 3-788. rtc_bkp_zoneb_sec_pro_check.....	525
Table 3-789. Function rtc_flag_get.....	525
Table 3-790. Function rtc_flag_clear.....	526
Table 3-791. Function rtc_interrupt_enable .....	527
Table 3-792. Function rtc_interrupt_disable .....	527
Table 3-793. Function rtc_nsec_interrupt_flag_get .....	528
Table 3-794. Function rtc_nsec_interrupt_flag_clear .....	529
Table 3-795. Function rtc_sec_interrupt_flag_get.....	529
Table 3-796. Function rtc_nsec_interrupt_flag_clear .....	530
Table 3-797. SDIO Registers .....	531
Table 3-798. SDIO firmware function .....	532
Table 3-799. Function sdio_deinit .....	533
Table 3-800. Function sdio_clock_config.....	534
Table 3-801. Function sdio_hardware_clock_enable.....	535
Table 3-802. Function sdio_hardware_clock_disable.....	535
Table 3-803. Function sdio_bus_mode_set .....	536
Table 3-804. Function sdio_power_state_set .....	536
Table 3-805. Function sdio_power_state_get .....	537
Table 3-806. Function sdio_clock_enable .....	537
Table 3-807. Function sdio_clock_disable.....	538
Table 3-808. Function sdio_command_response_config.....	538
Table 3-809. Function sdio_wait_type_set.....	539
Table 3-810. Function sdio_csm_enable .....	540
Table 3-811. Function sdio_csm_disable .....	540
Table 3-812. Function sdio_command_index_get.....	541
Table 3-813. Function sdio_response_get .....	541
Table 3-814. Function sdio_data_config .....	542
Table 3-815. Function sdio_data_transfer_config.....	543
Table 3-816. Function sdio_dsm_enable.....	544
Table 3-817. Function sdio_dsm_disable.....	544
Table 3-818. Function sdio_data_write.....	545
Table 3-819. Function sdio_data_read.....	545
Table 3-820. Function sdio_data_counter_get .....	546
Table 3-821. Function sdio_data_counter_get .....	546
Table 3-822. Function sdio_dma_enable.....	547
Table 3-823. Function sdio_dma_disable.....	547
Table 3-824. Function sdio_flag_get.....	548
Table 3-825. Function sdio_flag_clear.....	549
Table 3-826. Function sdio_interrupt_enable .....	550
Table 3-827. Function sdio_interrupt_disable .....	551
Table 3-828. Function sdio_interrupt_flag_get.....	552
Table 3-829. Function sdio_interrupt_flag_clear .....	554
Table 3-830. Function sdio_readwait_enable .....	555
Table 3-831. Function sdio_readwait_disable .....	555

Table 3-832. Function <code>sdio_stop_readwait_enable</code> .....	556
Table 3-833. Function <code>sdio_stop_readwait_disable</code> .....	556
Table 3-834. Function <code>sdio_readwait_type_set</code> .....	557
Table 3-835. Function <code>sdio_operation_enable</code> .....	557
Table 3-836. Function <code>sdio_operation_disable</code> .....	558
Table 3-837. Function <code>sdio_suspend_enable</code> .....	558
Table 3-838. Function <code>sdio_suspend_disable</code> .....	559
Table 3-839. Function <code>sdio_ceata_command_enable</code> .....	559
Table 3-840. Function <code>sdio_ceata_command_disable</code> .....	560
Table 3-841. Function <code>sdio_ceata_interrupt_enable</code> .....	560
Table 3-842. Function <code>sdio_ceata_interrupt_disable</code> .....	561
Table 3-843. Function <code>sdio_ceata_command_completion_enable</code> .....	561
Table 3-844. Function <code>sdio_ceata_command_completion_disable</code> .....	562
Table 3-845. SPI/I2S Registers .....	562
Table 3-846. SPI/I2S firmware function .....	563
Table 3-847. Structure <code>spi_parameter_struct</code> .....	564
Table 3-848. Function <code>spi_i2s_deinit</code> .....	564
Table 3-849. Function <code>spi_struct_para_init</code> .....	565
Table 3-850. Function <code>spi_init</code> .....	565
Table 3-851. Function <code>spi_enable</code> .....	566
Table 3-852. Function <code>spi_disable</code> .....	567
Table 3-853. Function <code>i2s_init</code> .....	567
Table 3-854. Function <code>i2s_psc_config</code> .....	568
Table 3-855. Function <code>i2s1_ckin_psc_config</code> .....	569
Table 3-856. Function <code>i2s_enable</code> .....	571
Table 3-857. Function <code>i2s_disable</code> .....	571
Table 3-858. Function <code>spi_nss_output_enable</code> .....	572
Table 3-859. Function <code>spi_nss_output_disable</code> .....	572
Table 3-860. Function <code>spi_nss_internal_high</code> .....	573
Table 3-861. Function <code>spi_nss_internal_low</code> .....	573
Table 3-862. Function <code>spi_dma_enable</code> .....	574
Table 3-863. Function <code>spi_dma_disable</code> .....	574
Table 3-864. Function <code>spi_i2s_data_frame_format_config</code> .....	575
Table 3-865. Function <code>spi_i2s_data_transmit</code> .....	576
Table 3-866. Function <code>spi_i2s_data_receive</code> .....	576
Table 3-867. Function <code>spi_bidirectional_transfer_config</code> .....	577
Table 3-868. Function <code>i2s_init</code> .....	577
Table 3-869. Function <code>spi_i2s_format_error_clear</code> .....	579
Table 3-870. Function <code>spi_crc_polynomial_set</code> .....	579
Table 3-871. Function <code>spi_crc_polynomial_get</code> .....	580
Table 3-872. Function <code>spi_crc_on</code> .....	580
Table 3-873. Function <code>spi_crc_off</code> .....	581
Table 3-874. Function <code>spi_crc_next</code> .....	581
Table 3-875. Function <code>spi_crc_get</code> .....	582

Table 3-876. Function spi_crc_error_clear.....	583
Table 3-877. Function spi_ti_mode_enable.....	583
Table 3-878. Function spi_ti_mode_disable.....	584
Table 3-879. Function spi_quad_enable.....	584
Table 3-880. Function spi_quad_disable.....	585
Table 3-881. Function spi_quad_write_enable .....	585
Table 3-882. Function spi_quad_read_enable .....	586
Table 3-883. Function spi_i2s_flag_get.....	586
Table 3-884. Function spi_i2s_interrupt_enable .....	587
Table 3-885. Function spi_i2s_interrupt_disable .....	588
Table 3-886. Function spi_i2s_interrupt_flag_get .....	588
Table 3-887. SQPI Registers .....	589
Table 3-888. SQPI firmware function .....	590
Table 3-889. sqpi_parameter_struct .....	590
Table 3-890. Function sqpi_deinit .....	590
Table 3-891. Function sqpi_struct_para_init .....	591
Table 3-892. Function sqpi_init .....	591
Table 3-893. Function sqpi_read_id_command.....	592
Table 3-894. Function sqpi_special_command .....	593
Table 3-895. Function sqpi_read_command_config .....	593
Table 3-896. Function sqpi_write_command_config .....	594
Table 3-897. Function sqpi_low_id_receive.....	595
Table 3-898. Function sqpi_low_id_receive.....	595
Table 3-899. SYSCFG registers .....	596
Table 3-900. SYSCFG firmware function .....	596
Table 3-901. Function syscfg_deinit .....	597
Table 3-902. Function syscfg_exti_line_config .....	598
Table 3-903. Function syscfg_compensation_pwdn_mode_enable .....	598
Table 3-904. Function syscfg_compensation_pwdn_mode_disable .....	599
Table 3-905. Function syscfg_clock_access_security_config .....	599
Table 3-906. Function syscfg_classb_access_security_config .....	600
Table 3-907. Function syscfg_sram1_access_security_config .....	600
Table 3-908. Function syscfg_fpu_access_security_config .....	601
Table 3-909. Function syscfg_vtor_ns_write_disable.....	602
Table 3-910. Function syscfg_mpu_ns_write_disable.....	602
Table 3-911. Function syscfg_vtors_aircr_write_disable .....	603
Table 3-912. Function syscfg_mpu_s_write_disable .....	603
Table 3-913. Function sau_write_disable.....	604
Table 3-914. Function syscfg_lock_config.....	604
Table 3-915. Function syscfg_gssacmd_write_data .....	605
Table 3-916. Function syscfg_sram1_erase .....	605
Table 3-917. Function syscfg_sram1_unlock .....	606
Table 3-918. Function syscfg_sram1_lock.....	606
Table 3-919. Function syscfg_sram1_write_protect_0_31 .....	607

Table 3-920. Function syscfg_sram1_write_protect_32_63 .....	607
Table 3-921. Function syscfg_compensation_ready_flag_get .....	608
Table 3-922. Function syscfg_sram1_bsy_flag_get.....	608
Table 3-923. Function syscfg_fpu_interrupt_enable.....	609
Table 3-924. Function syscfg_fpu_interrupt_disable.....	609
Table 3-925. TIMEx Registers.....	610
Table 3-926. TIMEx firmware function.....	611
Table 3-927. Structure timer_parameter_struct.....	613
Table 3-928. Structure timer_break_parameter_struct .....	614
Table 3-929. Structure timer_oc_parameter_struct.....	614
Table 3-930. Structure timer_ic_parameter_struct.....	614
Table 3-931. Function timer_deinit.....	615
Table 3-932. Function timer_struct_para_init .....	615
Table 3-933. Function timer_init.....	616
Table 3-934. Function timer_enable.....	616
Table 3-935. Function timer_disable.....	617
Table 3-936. Function timer_auto_reload_shadow_enable .....	617
Table 3-937. Function timer_auto_reload_shadow_disable.....	618
Table 3-938. Function timer_update_event_enable .....	618
Table 3-939. Function timer_update_event_disable .....	619
Table 3-940. Function timer_counter_alignment .....	619
Table 3-941. Function timer_counter_up_direction .....	620
Table 3-942. timer_counter_down_direction .....	621
Table 3-943. Function timer_prescaler_config .....	621
Table 3-944. Function timer_repetition_value_config.....	622
Table 3-945. Function timer_autoreload_value_config.....	623
Table 3-946. Function timer_counter_value_config.....	623
Table 3-947. Function timer_counter_read .....	624
Table 3-948. Function timer_prescaler_read .....	624
Table 3-949. Function timer_single_pulse_mode_config.....	625
Table 3-950. Function timer_update_source_config.....	625
Table 3-951. Function timer_dma_enable .....	626
Table 3-952. Function timer_dma_disable .....	627
Table 3-953. Function timer_channel_dma_request_source_select.....	628
Table 3-954. Function timer_dma_transfer_config .....	628
Table 3-955. Function timer_event_software_generate.....	630
Table 3-956. Function timer_break_struct_para_init .....	631
Table 3-957. Function timer_break_config.....	632
Table 3-958. Function timer_break_enable .....	632
Table 3-959. Function timer_break_disable .....	633
Table 3-960. Function timer_automatic_output_enable .....	633
Table 3-961. Function timer_automatic_output_disable .....	634
Table 3-962. Function timer_primary_output_config.....	635
Table 3-963. Function timer_channel_control_shadow_config.....	635



Table 3-964. Function timer_channel_control_shadow_update_config .....	636
Table 3-965. Function timer_channel_output_struct_para_init .....	636
Table 3-966. Function timer_channel_output_config .....	637
Table 3-967. Function timer_channel_output_mode_config .....	638
Table 3-968. Function timer_channel_output_pulse_value_config .....	639
Table 3-969. Function timer_channel_output_shadow_config .....	640
Table 3-970. Function timer_channel_output_fast_config .....	641
Table 3-971. Function timer_channel_output_clear_config .....	641
Table 3-972. Function timer_channel_output_polarity_config .....	642
Table 3-973. Function timer_channel_complementary_output_polarity_config .....	643
Table 3-974. Function timer_channel_output_state_config .....	644
Table 3-975. Function timer_channel_complementary_output_state_config .....	645
Table 3-976. Function timer_channel_input_struct_para_init .....	645
Table 3-977. Function timer_input_capture_config .....	646
Table 3-978. Function timer_channel_input_capture_prescaler_config .....	647
Table 3-979. Function timer_channel_capture_value_register_read .....	648
Table 3-980. Function timer_input_pwm_capture_config .....	648
Table 3-981. Function timer_hall_mode_config .....	649
Table 3-982. Function timer_input_trigger_source_select .....	650
Table 3-983. Function timer_master_output_trigger_source_select .....	651
Table 3-984. Function timer_slave_mode_select .....	652
Table 3-985. Function timer_master_slave_mode_config .....	653
Table 3-986. Function timer_external_trigger_config .....	654
Table 3-987. Function timer_quadrature_decoder_mode_config .....	655
Table 3-988. Function timer_internal_clock_config .....	656
Table 3-989. Function timer_internal_trigger_as_external_clock_config .....	656
Table 3-990. Function timer_external_trigger_as_external_clock_config .....	657
Table 3-991. Function timer_external_clock_mode0_config .....	658
Table 3-992. Function timer_external_clock_mode1_config .....	659
Table 3-993. Function timer_external_clock_mode1_disable .....	660
Table 3-994. Function timer_write_chxval_register_config .....	660
Table 3-995. Function timer_output_value_selection_config .....	661
Table 3-996. Function timer_flag_get .....	662
Table 3-997. Function timer_flag_clear .....	662
Table 3-998. Function timer_interrupt_enable .....	663
Table 3-999. Function timer_interrupt_disable .....	664
Table 3-1000. Function timer_interrupt_flag_get .....	665
Table 3-1001. Function timer_interrupt_flag_clear .....	666
Table 3-1002 TRNG Registers .....	667
Table 3-1003. TRNG firmware function .....	667
Table 3-1004. Enum trng_flag_enum .....	667
Table 3-1005. Enum trng_int_flag_enum .....	667
Table 3-1006. Function trng_deinit .....	667
Table 3-1007. Function trng_enable .....	668

Table 3-1008 Function trng_disable.....	668
Table 3-1009. Function trng_powermode_config.....	669
Table 3-1010 Function trng_get_true_random_data .....	670
Table 3-1011 trng_interrupt_enable .....	670
Table 3-1012 trng_interrupt_disable .....	671
Table 3-1013 trng_flag_get .....	671
Table 3-1014 trng_interrupt_flag_get.....	672
Table 3-1015 trng_interrupt_flag_clear.....	672
Table 3-1016. TSI Registers .....	673
Table 3-1017. TSI firmware function .....	673
Table 3-1018. Function tsi_deinit .....	674
Table 3-1019. Function tsi_init .....	675
Table 3-1020. Function tsi_enable .....	676
Table 3-1021. Function tsi_disable .....	677
Table 3-1022. Function tsi_sample_pin_enable .....	677
Table 3-1023. Function tsi_sample_pin_disable .....	678
Table 3-1024. Function tsi_channel_pin_enable .....	678
Table 3-1025. Function tsi_channel_pin_disable .....	679
Table 3-1026. Function tsi_software_mode_config .....	679
Table 3-1027. Function tsi_software_start .....	680
Table 3-1028. Function tsi_software_stop .....	680
Table 3-1029. Function tsi_hardware_mode_config .....	681
Table 3-1030. Function tsi_pin_mode_config .....	681
Table 3-1031. Function tsi_extend_charge_config .....	682
Table 3-1032. Function tsi_plus_config.....	683
Table 3-1033. Function tsi_max_number_config .....	684
Table 3-1034. Function tsi_hysteresis_on .....	684
Table 3-1035. Function tsi_hysteresis_off .....	685
Table 3-1036. Function tsi_analog_on.....	685
Table 3-1037. Function tsi_analog_off.....	686
Table 3-1038. Function tsi_group_enable .....	686
Table 3-1039. Function tsi_group_disable .....	687
Table 3-1040. Function tsi_group_status_get .....	687
Table 3-1041. Function tsi_group0_cycle_get .....	688
Table 3-1042. Function tsi_group1_cycle_get .....	689
Table 3-1043. Function tsi_group2_cycle_get .....	689
Table 3-1044. Function tsi_flag_clear .....	690
Table 3-1045. Function tsi_flag_get .....	690
Table 3-1046. Function tsi_interrupt_enable .....	691
Table 3-1047. Function tsi_interrupt_disable .....	691
Table 3-1048. Function tsi_interrupt_flag_clear .....	692
Table 3-1049. Function tsi_interrupt_flag_get .....	692
Table 3-1050. TZPCU Registers .....	693
Table 3-1051. TZPCU firmware function .....	694

Table 3-1052. tzpcu_mem .....	695
Table 3-1053. tzpcu_non_secure_mark_region .....	695
Table 3-1054. tzpcu_non_secure_mark_struct .....	695
Table 3-1055. Function tzpcu_tzspc_peripheral_attributes_config .....	695
Table 3-1056. Function tzpcu_tzspc_peripheral_attributes_get .....	697
Table 3-1057. Function tzpcu_non_secure_mark_struct_para_init .....	699
Table 3-1058. Function tzpcu_tzspc_emnsm_config .....	700
Table 3-1059. Function tzpcu_tzspc_items_lock .....	700
Table 3-1060. Function tzpcu_tzspc_dbg_config .....	701
Table 3-1061. Function tzpcu_tzbmpc_lock .....	702
Table 3-1062. Function tzpcu_tzbmpc_security_state_config .....	702
Table 3-1063. Function tzpcu_tzbmpc_secure_access_config .....	703
Table 3-1064. Function tzpcu_tzbmpc_block_secure_access_mode_config .....	703
Table 3-1065. Function tzpcu_tzbmpc_union_block_lock .....	704
Table 3-1066. Function tzpcu_tziac_interrupt_enable .....	705
Table 3-1067. Function tzpcu_tziac_interrupt_disable .....	708
Table 3-1068. Function tzpcu_tziac_flag_get .....	710
Table 3-1069. Function tzpcu_tziac_flag_clear .....	713
Table 3-1070. USART Registers .....	717
Table 3-1071. USART firmware function .....	717
Table 3-1072. Enum usart_flag_enum .....	719
Table 3-1073. Enum usart_interrupt_flag_enum .....	720
Table 3-1074. Enum usart_interrupt_enum .....	720
Table 3-1075. Enum usart_invert_enum .....	721
Table 3-1076. Function usart_deinit .....	721
Table 3-1077. Function usart_baudrate_set .....	722
Table 3-1078. Function usart_parity_config .....	722
Table 3-1079. Function usart_word_length_set .....	723
Table 3-1080. Function usart_stop_bit_set .....	723
Table 3-1081. Function usart_enable .....	724
Table 3-1082. Function usart_disable .....	725
Table 3-1083. Function usart_transmit_config .....	725
Table 3-1084. Function usart_receive_config .....	726
Table 3-1085. Function usart_data_first_config .....	726
Table 3-1086. Function usart_invert_config .....	727
Table 3-1087. Function usart_overrun_enable .....	728
Table 3-1088. Function usart_overrun_enable .....	728
Table 3-1089. Function usart_oversample_config .....	729
Table 3-1090. Function usart_sample_bit_config .....	730
Table 3-1091. Function usart_receiver_timeout_enable .....	730
Table 3-1092. Function usart_receiver_timeout_disable .....	731
Table 3-1093. Function usart_receiver_timeout_threshold_config .....	731
Table 3-1094. Function usart_data_transmit .....	732
Table 3-1095. Function usart_data_receive .....	732

Table 3-1096. Function <code>usart_address_config</code> .....	733
Table 3-1097. Function <code>usart_address_detection_mode_config</code> .....	733
Table 3-1098. Function <code>usart_mute_mode_enable</code> .....	734
Table 3-1099. Function <code>usart_mute_mode_disable</code> .....	734
Table 3-1100. Function <code>usart_mute_mode_wakeup_config</code> .....	735
Table 3-1101. Function <code>usart_lin_mode_enable</code> .....	736
Table 3-1102. Function <code>usart_lin_mode_disable</code> .....	736
Table 3-1103. Function <code>usart_lin_break_dection_length_config</code> .....	737
Table 3-1104. Function <code>usart_halfduplex_enable</code> .....	737
Table 3-1105. Function <code>usart_halfduplex_disable</code> .....	738
Table 3-1106. Function <code>usart_clock_enable</code> .....	738
Table 3-1107. Function <code>usart_clock_disable</code> .....	739
Table 3-1108. Function <code>usart_synchronous_clock_config</code> .....	739
Table 3-1109. Function <code>usart_guard_time_config</code> .....	740
Table 3-1110. Function <code>usart_smartcard_mode_enable</code> .....	741
Table 3-1111. Function <code>usart_smartcard_mode_disable</code> .....	741
Table 3-1112. Function <code>usart_smartcard_mode_nack_enable</code> .....	742
Table 3-1113. Function <code>usart_smartcard_mode_nack_disable</code> .....	742
Table 3-1114. Function <code>usart_smartcard_mode_early_nack_enable</code> .....	743
Table 3-1115. Function <code>usart_smartcard_mode_early_nack_disable</code> .....	743
Table 3-1116. Function <code>usart_smartcard_autoretry_config</code> .....	744
Table 3-1117. Function <code>usart_block_length_config</code> .....	744
Table 3-1118. Function <code>usart_irda_mode_enable</code> .....	745
Table 3-1119. Function <code>usart_irda_mode_disable</code> .....	745
Table 3-1120. Function <code>usart_prescaler_config</code> .....	746
Table 3-1121. Function <code>usart_irda_lowpower_config</code> .....	746
Table 3-1122. Function <code>usart_hardware_flow_rts_config</code> .....	747
Table 3-1123. Function <code>usart_hardware_flow_cts_config</code> .....	748
Table 3-1124. Function <code>usart_hardware_flow_coherence_config</code> .....	748
Table 3-1125. Function <code>usart_rs45_driver_enable</code> .....	749
Table 3-1126. Function <code>usart_rs45_driver_disable</code> .....	749
Table 3-1127. Function <code>usart_driver_asserttime_config</code> .....	750
Table 3-1128. Function <code>usart_driver_deasserttime_config</code> .....	750
Table 3-1129. Function <code>usart_depolarity_config</code> .....	751
Table 3-1130. Function <code>usart_dma_receive_config</code> .....	752
Table 3-1131. Function <code>usart_dma_transmit_config</code> .....	752
Table 3-1132. Function <code>usart_reception_error_dma_disable</code> .....	753
Table 3-1133. Function <code>usart_reception_error_dma_enable</code> .....	753
Table 3-1134. Function <code>usart_wakeup_enable</code> .....	754
Table 3-1135. Function <code>usart_wakeup_disable</code> .....	754
Table 3-1136. Function <code>usart_wakeup_mode_config</code> .....	755
Table 3-1137. Function <code>usart_receive_fifo_enable</code> .....	755
Table 3-1138. Function <code>usart_receive_fifo_disable</code> .....	756
Table 3-1139. Function <code>usart_receive_fifo_counter_number</code> .....	756

Table 3-1140. Function usart_command_enable .....	757
Table 3-1141. Function usart_flag_get.....	758
Table 3-1142. Function usart_flag_clear.....	759
Table 3-1143. Function usart_interrupt_enable .....	760
Table 3-1144. Function usart_interrupt_disable .....	761
Table 3-1145. Function usart_interrupt_flag_get.....	762
Table 3-1146. Function usart_interrupt_flag_clear.....	763
Table 3-1147. WWDGT Registers.....	764
Table 3-1148. WWDGT firmware function.....	765
Table 3-1149. Function wwdgt_deinit .....	765
Table 3-1150. Function wwdgt_enable.....	765
Table 3-1151. Function wwdgt_counter_update.....	766
Table 3-1152. Function wwdgt_config .....	766
Table 3-1153. Function wwdgt_interrupt_enable.....	767
Table 3-1154. Function wwdgt_flag_get .....	768
Table 3-1155. Function wwdgt_flag_clear .....	768
Table 4-1. Revision history .....	769

## 1. Introduction

This manual introduces firmware library of GD32W51x\_F5HC 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 GD32W51x\_F5HC 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
CAU	Cryptographic acceleration unit
CRC	CRC calculation unit
CTC	Clock trim controller
DBG	Debug

Peripherals	Descriptions
DCI	Digital camera interface
DMA	Direct memory access controller
EFUSE	Electronic fuse
EXTI	Interrupt/event controller
FMC	Flash memory controller
FWDGT	Free watchdog timer
GPIO	General-purpose and alternate-function I/Os
HAU	Hash acceleration unit
HPDF	High-performance digital filter
I2C	Inter-integrated circuit interface
ICACHE	Instruction cache
IFRP	Infrared ray port
MISC	Nested Vectored Interrupt Controller
PKCAU	Public key cryptographic acceleration unit
PMU	Power management unit
QSPI	Quad-SPI interface
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
TRNG	True random number generator
TSI	Touch sensing interface
TZPCU	TrustZone protection controller union
USART	Universal synchronous/asynchronous receiver /transmitter
WWDGT	Window watchdog timer
USBFS	Universal serial bus full-speed interface

## 1.1.2. Naming rules

The firmware library naming rules are shown as below:

- The peripherals are shortened in XXX format, such as: ADC. More shorten information of peripherals refer to [Peripherals](#);
- The name of sourcefile and header file are started with “gd32w51x\_f5hc\_”, such as: gd32w51x\_f5hc\_adc.h;
- The constants used only in one file should be defined in the used file; the constants used in many files should be defined in corresponding header file. All the constants are written in uppcase of English letters;
- Registers are handled as constants. The naming of them are written in uppcase of English letters. In most cases, register names are shortened accord with the user manual;

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

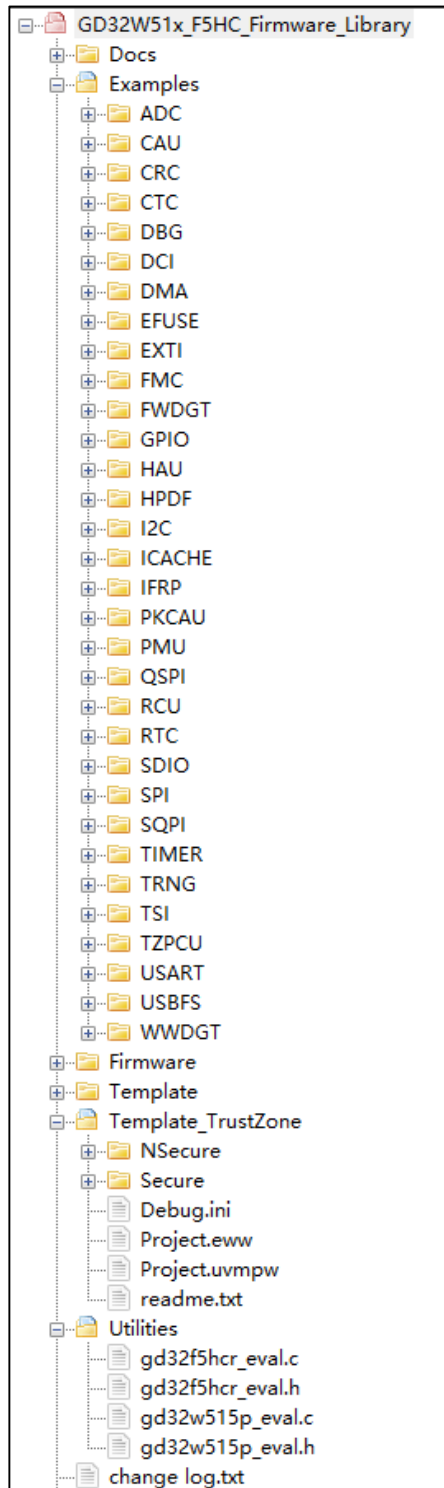


## 2. Firmware Library Overview

### 2.1. File Structure of Firmware Library

GD32W51x\_F5HC\_Firmware\_Library, the file structure is shown as below:

**Figure 2-1. File structure of firmware library of GD32W51x\_F5HC**



### 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;
- `gd32w51x_f5hc_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);
- `gd32w51x_f5hc_it.c`: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- `gd32w51x_f5hc_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.

The GD32W51x\_F5HC series MCU based on Arm® Cortex™-M33 core and supports Arm® Trustzone® technology. Some peripherals include trustzone enabling examples. In these examples, the following files will be added:

- `partition_gd32w51x_f5hc.h`: the header file configures SAU/IDAU and secure/non-secure interrupts attributes;
- `Project_S.sct`: the keil scatter-loading file provides detailed information about the grouping and placement of each area and part of the image in the security project;
- `Project_NS.sct`: the keil scatter-loading file provides detailed information about the grouping and placement of each area and part of the image in the non-security project;
- `gd32w51x_flash_s.icf` / `gd32f5hc_flash_s.icf`: the IAR linker configuration file provides to link and locate an application in memory according requirements in the security project;
- `gd32w51x_flash_ns.icf` / `gd32f5hc_flash_ns.icf`: the IAR linker configuration file provides to link and locate an application in memory according requirements in the non-security project;
- `gd32w51x_flash_s.ld` / `gd32f5hc_flash_s.ld`: In secure projects, the EBuilder linker script files link and load various parts of the program into specified memory regions as required;
- `gd32w51x_flash_ns.ld` / `gd32f5hc_flash_ns.ld`: In non-secure projects, the EBuilder linker script files link and load various parts of the program into specified memory regions as required.

### 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 GD32W51x\_F5HC and system

configuration file;

- GD32W51x\_F5HC\_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;
- GD32W51x\_F5HC\_usbfs\_library subfolder includes all the related files about USBFS peripheral:

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

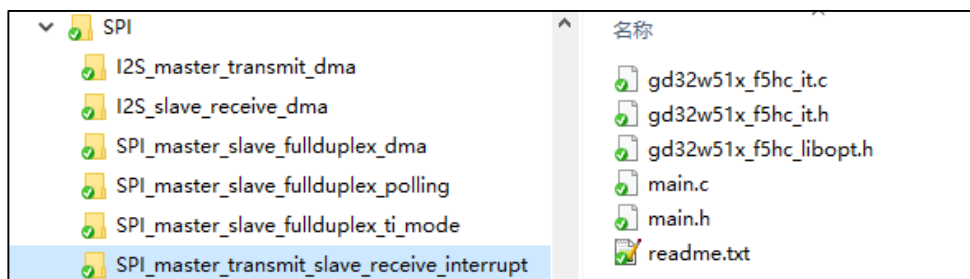
## 2.1.3. Template Folder

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, Keil\_project is run in Keil5, and GD32EBuilder\_project is used for the GD32EBuilder). 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:

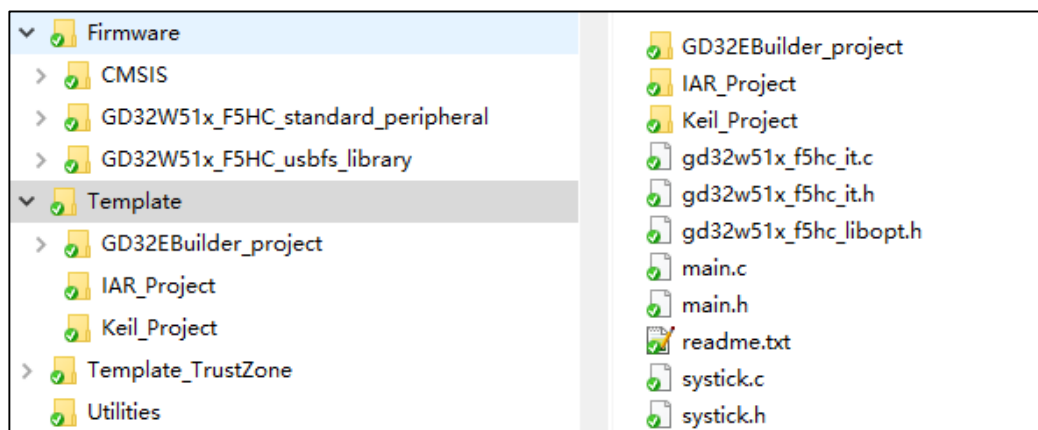
**Figure 2-2. Select peripheral example files**



### 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:

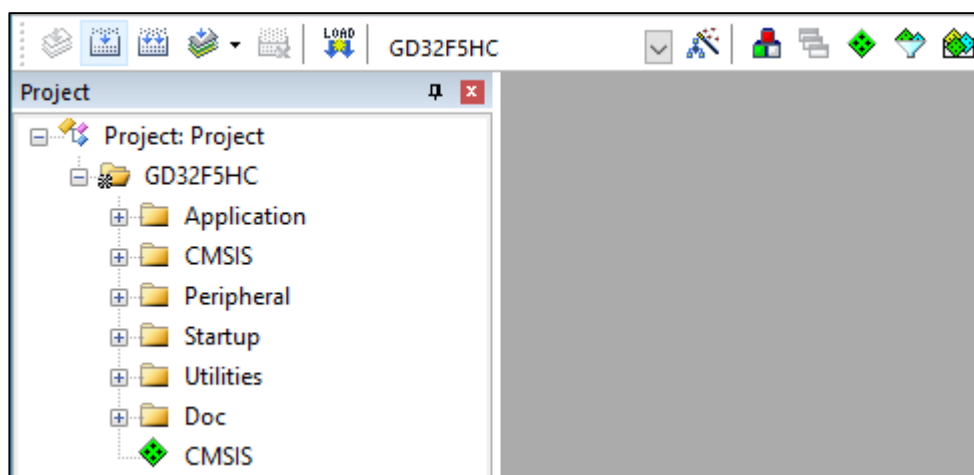
**Figure 2-3. Copy the peripheral example files**



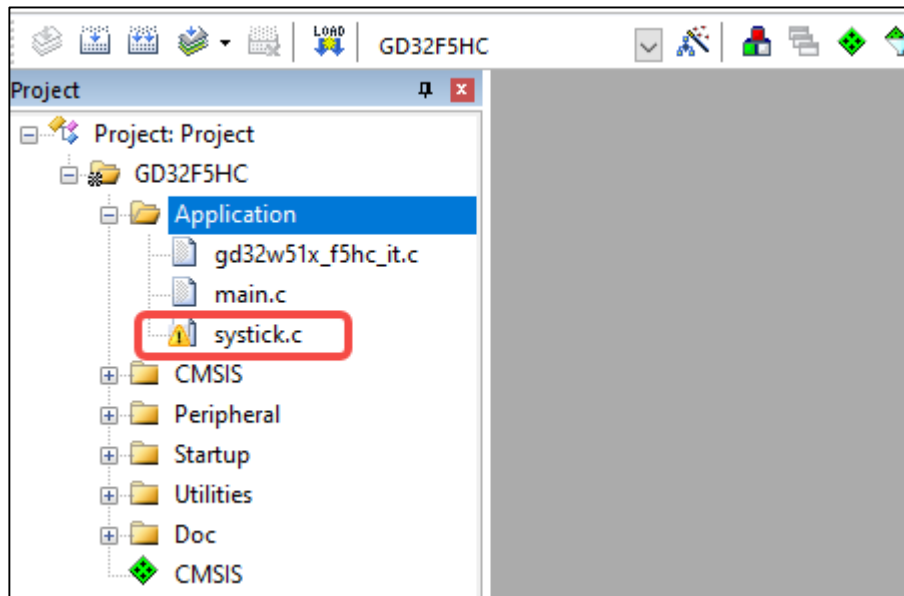
### Open a project

GD provides project in Keil, IAR and GD32EBuilder, 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**

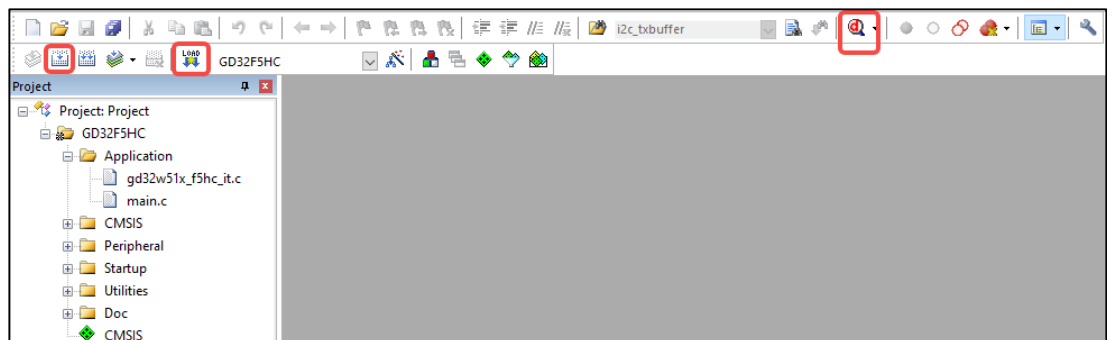


Because different module and different functions adopt different files, users should add or delete the files in project according to the copied files, shown as below:

**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**

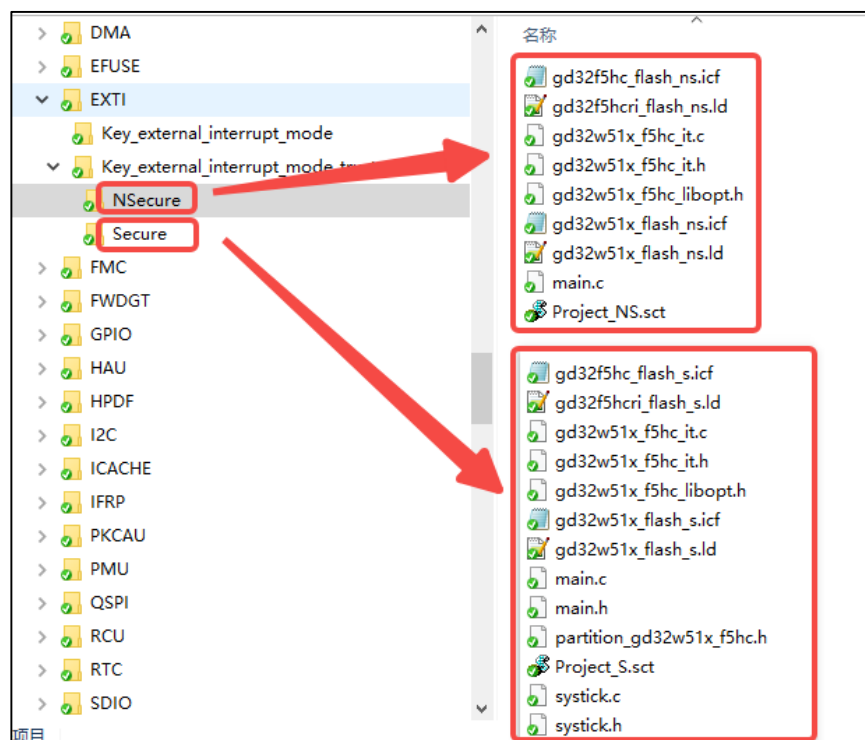
#### 2.1.4. Template\_TrustZone Folder

Template\_TrustZone folder includes two subdirectories, Secure and Nsecure. A simple demo of how to switch between secure application and non-secure application, (IAR\_project is run in IAR, Keil\_project is run in Keil5, and GD32EBuilder\_project is used for the GD32EBuilder). 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 EXTI, open “EXTI” folder, select an trustzone example of EXTI, such as “Key\_external\_interrupt\_mode\_trustzone”, shown as below:

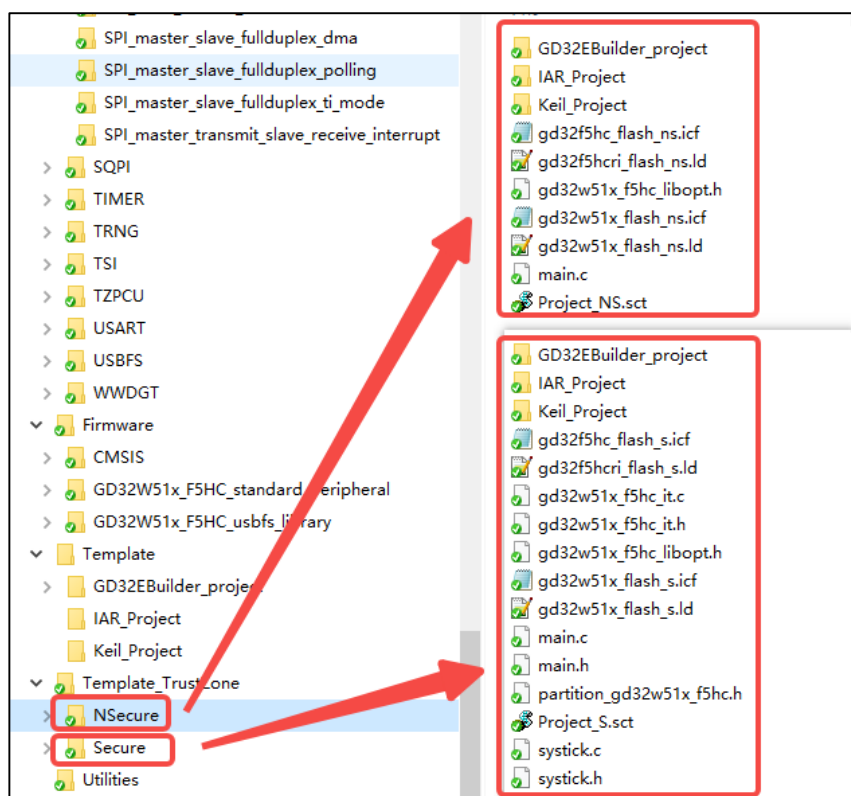
**Figure 2-7. Select peripheral example files**



## Copy files

Open the "Template\_TrustZone" folder, keep the "IAR\_project," "Keil\_project," and "GD32EBuilder\_project" subfolders in the Secure and NSecure folders, and delete all other files, then copy all the files in “Key\_external\_interrupt\_mode\_trustzone” folder to the “Template\_Trustzone” subfolder secure and nsecure separately, shown as below:

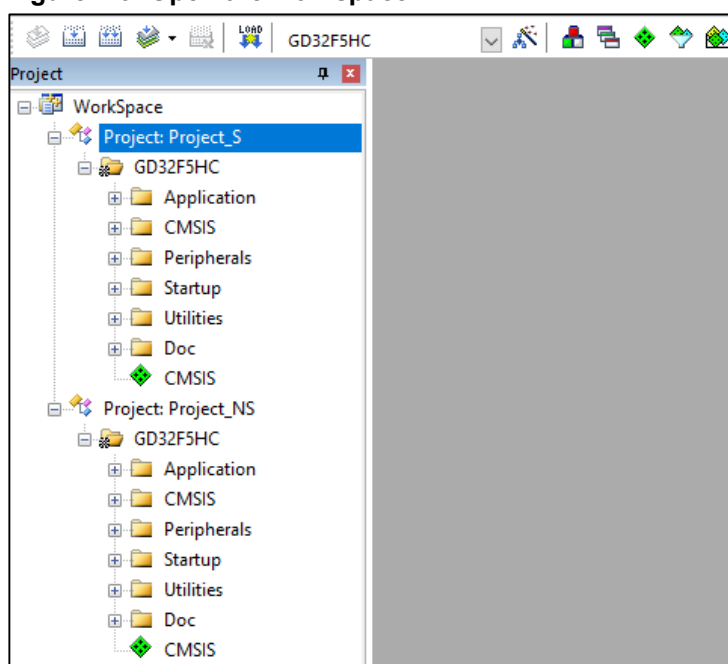
**Figure 2-8. Copy the peripheral example files**



## Open a workspace

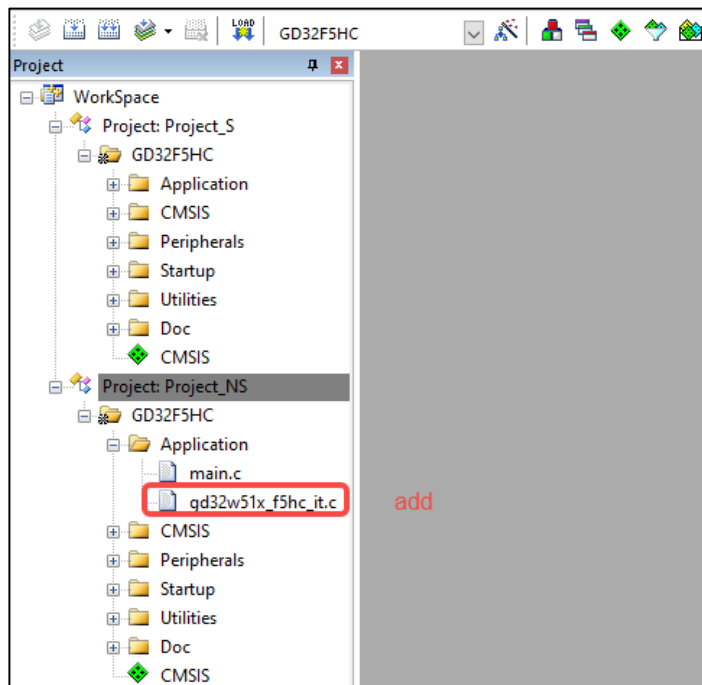
GD provides workspace in Keil and IAR, users can open workspace in different IDEs according to their need, such as "Keil\_project" workspace, open \ Template\_TrustZone\ Project.uvmpw, shown as below:

**Figure 2-9. Open the workspace**



The workspace include two projects, secure project and non-secure project. 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:

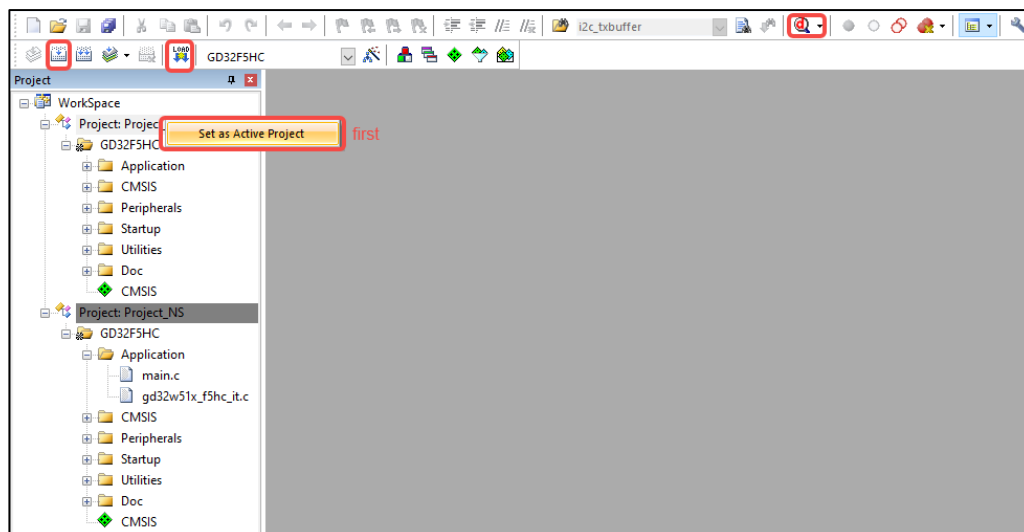
**Figure 2-10. Configure project files**



## Compile-Debug-Download

First compile two projects separately, if there is no error, then select the right jumper cap according to the description of readme, download the two projects to the target board separately, 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 2-11. Compile-debug-download**





## 2.1.5. Utilities Folder

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

- gd32f5hcr\_eval.h is the header file required for running firmware library examples related to the GD32F5HCR\_EVAL evaluation board.
- gd32f5hcr\_eval.c is the source file required for running firmware library examples related to the GD32F5HCR\_EVAL evaluation board.
- gd32w515p\_eval.h is the header file required for running firmware library examples related to the GD32W515P\_EVAL evaluation board.
- gd32w515p\_eval.c is the source file required for running firmware library examples related to the GD32W515P\_EVAL evaluation board.

**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
gd32w51x_f5hc_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.
gd32w51x_f5hc_it.h	Header file, including all the prototypes of interrupt service routines.
gd32w51x_f5hc_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.
gd32w51x_f5hc_xxx.h	The header file of peripheral PPP, including functions about peripheral PPP, and the variables used for functions.
gd32w51x_f5hc_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.
partition_gd32w51x_f5hc.h	The header file configures SAU/IDAU and secure/non-secure interrupts attributes;

## 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**

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

<b>Registers</b>	<b>Descriptions</b>
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_WD0HT	Watchdog 0 high threshold register
ADC_WD0LT	Watchdog 0 low threshold register

Registers	Descriptions
ADC_RSQ0	Routine sequence register 0
ADC_RSQ1	Routine sequence register 1
ADC_RSQ2	Routine sequence register 2
ADC_ISQ	Inserted sequence register
ADC_IDATAx (x=0..3)	Inserted data register x(x=0..3)
ADC_LDATAx (x=0..3)	Latch data register x(x=0..3)
ADC_RDATA	Routine data register
ADC_IDATAx	Inserted data register
ADC_LDCTL	Latch data control register
ADC_OVSAMPCTL	Oversample control register
ADC_CCTL	Common 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_clock_config	configure the ADC clock
adc_enable	enable ADC interface
adc_disable	disable ADC interface
adc_dma_mode_enable	enable DMA request
adc_dma_mode_disable	disable DMA request
adc_dma_request_after_last_enable	when DMA=1, the DMA engine issues a request at end of each conversion
adc_dma_request_after_last_disable	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
adc_end_of_conversion_config	configure end of conversion mode
adc_internal_channel_config	enable or disable ADC internal channels
adc_discontinuous_mode_config	configure ADC discontinuous mode
adc_special_function_config	enable or disable ADC special function
adc_data_alignment_config	configure ADC data alignment
adc_channel_length_config	configure the length of routine channel sequence or inserted channel sequence
adc_routine_channel_config	configure ADC routine channel
adc_inserted_channel_config	configure ADC inserted channel
adc_inserted_channel_offset_config	configure ADC inserted channel offset
adc_external_trigger_config	enable ADC external trigger
adc_external_trigger_source_config	configure ADC external trigger source

Function name	Function description
adc_software_trigger_enable	enable ADC software trigger
adc_routine_data_read	read ADC routine sequence data register
adc_inserted_data_read	read ADC inserted sequence data register
adc_inserted_data_read	read ADC inserted sequence data register
adc_latch_data_read	read ADC latch data register
adc_latch_data_source_config	configure ADC latch data source
adc_watchdog0_single_channel_enable	configure ADC analog watchdog 0 single channel
adc_watchdog0_sequence_channel_enable	configure ADC analog watchdog 0 sequence channel
adc_watchdog0_disable	disable ADC analog watchdog 0
adc_watchdog0_threshold_config	configure ADC analog watchdog 0 threshold
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(void);
Function descriptions	reset ADC peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

## adc\_clock\_config

The description of adc\_clock\_config is shown as below:

**Table 3-5. Function adc\_clock\_config**

<b>Function name</b>	adc_clock_config
<b>Function prototype</b>	void adc_clock_config(uint32_t prescaler);
<b>Function descriptions</b>	configure the ADC clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>prescaler</b>	configure ADC prescaler ratio
ADC_ADCCCK_PCLK2_DIV2	PCLK2 div2
ADC_ADCCCK_PCLK2_DIV4	PCLK2 div4
ADC_ADCCCK_PCLK2_DIV6	PCLK2 div6
ADC_ADCCCK_PCLK2_DIV8	PCLK2 div8
ADC_ADCCCK_HCLK_DIV5	HCLK div5
ADC_ADCCCK_HCLK_DIV6	HCLK div6
ADC_ADCCCK_HCLK_DIV10	HCLK div10
ADC_ADCCCK_HCLK_DIV20	HCLK div20
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the ADC clock */
adc_clock_config(ADC_ADCCCK_PCLK2_DIV8);
```

## adc\_enable

The description of adc\_enable is shown as below:

**Table 3-6. Function adc\_enable**

<b>Function name</b>	adc_enable
<b>Function prototype</b>	void adc_enable(void);

<b>Function descriptions</b>	enable ADC interface
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC */
adc_enable();
```

### adc\_disable

The description of adc\_disable is shown as below:

**Table 3-7. Function adc\_disable**

<b>Function name</b>	adc_disable
<b>Function prototype</b>	void adc_disable(void);
<b>Function descriptions</b>	disable ADC interface
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### adc\_dma\_mode\_enable

The description of adc\_dma\_mode\_enable is shown as below:

**Table 3-8. Function adc\_dma\_mode\_enable**

<b>Function name</b>	adc_dma_mode_enable
<b>Function prototype</b>	void adc_dma_mode_enable(uint8_t adc_sequence);
<b>Function descriptions</b>	enable ADC DMA request
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
<i>ADC_ROUTINE_CHAN NEL</i>	routine sequence
<i>ADC_INSERTED_ CHANNEL</i>	inserted sequence
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC routine sequence DMA request */
```

```
adc_dma_mode_enable(ADC_ROUTINE_CHANNEL);
```

## adc\_dma\_mode\_disable

The description of adc\_dma\_mode\_disable is shown as below:

**Table 3-9. Function adc\_dma\_mode\_disable**

<b>Function name</b>	adc_dma_mode_disable
<b>Function prototype</b>	void adc_dma_mode_disable(uint8_t adc_sequence);
<b>Function descriptions</b>	disable ADC DMA request
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
<i>ADC_ROUTINE_CHAN NEL</i>	routine sequence
<i>ADC_INSERTED_ CHANNEL</i>	inserted sequence
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC inserted sequence DMA request */
```

```
adc_dma_mode_disable(ADC_ROUTINE_CHANNEL);
```

## adc\_dma\_request\_after\_last\_enable

The description of adc\_dma\_request\_after\_last\_enable is shown as below:

**Table 3-10. Function adc\_dma\_request\_after\_last\_enable**

<b>Function name</b>	adc_dma_request_after_last_enable
<b>Function prototype</b>	void adc_dma_request_after_last_enable(uint8_t adc_sequence);
<b>Function descriptions</b>	when DMA=1, the DMA engine issues a request at end of each conversion
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
ADC_ROUTINE_CHAN NEL	routine sequence
ADC_INSERTED_ CHANNEL	inserted sequence
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* when DMA=1, the DMA engine issues a request at end of each conversion */
```

```
adc_dma_request_after_last_enable(ADC_ROUTINE_CHANNEL);
```

## adc\_dma\_request\_after\_last\_disable

The description of adc\_dma\_request\_after\_last\_disable is shown as below:

**Table 3-11. Function adc\_dma\_request\_after\_last\_disable**

<b>Function name</b>	adc_dma_request_after_last_disable
<b>Function prototype</b>	void adc_dma_request_after_last_disable (uint8_t adc_sequence);
<b>Function descriptions</b>	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
ADC_ROUTINE_CHAN NEL	routine sequence
ADC_INSERTED_ CHANNEL	inserted sequence
<b>Output parameter{out}</b>	
-	-



Return value	
-	-

Example:

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

```
adc_dma_request_after_last_enable(ADC_ROUTINE_CHANNEL);
```

### adc\_end\_of\_conversion\_config

The description of adc\_end\_of\_conversion\_config is shown as below:

**Table 3-12. Function adc\_end\_of\_conversion\_config**

Function name	adc_end_of_conversion_config
Function prototype	void adc_end_of_conversion_config(uint8_t end_selection);
Function descriptions	configure end of conversion mode
Precondition	-
The called functions	-
Input parameter{in}	
end_selection	end of conversion mode
ADC_EORC_SET_SEQUENCE	only at the end of a sequence of routine conversions, the EOC bit is set. overflow detection is disabled unless RDMA=1.
ADC_EORC_SET_CONVERSION	at the end of each routine conversion, the EOC bit is set. Overflow is detected automatically.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure end of conversion mode */
```

```
adc_end_of_conversion_config(ADC_EORC_SET_SEQUENCE);
```

### adc\_internal\_channel\_config

The description of adc\_internal\_channel\_config is shown as below:

**Table 3-13. Function adc\_internal\_channel\_config**

Function name	adc_internal_channel_config
Function prototype	void adc_internal_channel_config(uint32_t internal_channel, ControlStatus newvalue);
Function descriptions	enable or disable ADC internal channel
Precondition	-
The called functions	-
Input parameter{in}	

<b>internal_channel</b>	the internal channels
<i>ADC_CHANNEL_INTERNAL_VBAT</i>	vbat channel
<i>ADC_CHANNEL_INTERNAL_VREFINT</i>	vrefint channel
<i>ADC_CHANNEL_INTERNAL_TEMPSENSOR</i>	temperature sensor channel
<i>ADC_CHANNEL_INTERNAL_TEMP_VREF</i>	vrefint and temperature sensor channel
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

/\* enable ADC high-precision temperature sensor channel \*/

adc\_internal\_channel\_config (ADC\_CHANNEL\_INTERNAL\_TEMPSENSOR, ENABLE);

### adc\_discontinuous\_mode\_config

The description of adc\_discontinuous\_mode\_config is shown as below:

**Table 3-14. Function adc\_discontinuous\_mode\_config**

<b>Function name</b>	adc_discontinuous_mode_config
<b>Function prototype</b>	void adc_discontinuous_mode_config(uint8_t adc_sequence, uint8_t length);
<b>Function descriptions</b>	configure ADC discontinuous mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
<i>ADC_ROUTINE_CHANNEL</i>	routine sequence
<i>ADC_INSERTED_CHANNEL</i>	inserted sequence
<i>ADC_CHANNEL_DISCONTINUOUS_DISABLE</i>	disable discontinuous mode of routine and inserted sequence
<b>Input parameter{in}</b>	
<b>length</b>	number of conversions in discontinuous mode, the number can be 1..8 for regular channel, the number has no effect for inserted channel
<b>Output parameter{out}</b>	

-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC discontinuous mode */
```

```
adc_discontinuous_mode_config(ADC_ROUTINE_CHANNEL, 6);
```

## adc\_special\_function\_config

The description of adc\_special\_function\_config is shown as below:

**Table 3-15. Function adc\_special\_function\_config**

<b>Function name</b>	adc_special_function_config
<b>Function prototype</b>	void adc_special_function_config(uint32_t function, ControlStatus newvalue);
<b>Function descriptions</b>	enable or disable ADC special function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>function</b>	the function to config
ADC_SCAN_MODE	scan mode select
ADC_INSERTED_CHANNEL_AUTO	inserted sequence convert automatically
ADC_CONTINUOUS_MODE	continuous mode select
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
ENABLE	enable function
DISABLE	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC scan mode */
```

```
adc_special_function_config(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`**

<b>Function name</b>	<code>adc_data_alignment_config</code>
<b>Function prototype</b>	<code>void adc_data_alignment_config(uint32_t data_alignment);</code>
<b>Function descriptions</b>	configure ADC data alignment
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data_alignment</b>	data alignment select
<code>ADC_DATAALIGN_RIGHT</code>	right alignment
<code>ADC_DATAALIGN_LEFT</code>	left alignment
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC data alignment */
adc_data_alignment_config(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`**

<b>Function name</b>	<code>adc_channel_length_config</code>
<b>Function prototype</b>	<code>void adc_channel_length_config(uint8_t adc_sequence, uint32_t length);</code>
<b>Function descriptions</b>	configure the length of routine sequence or inserted sequence
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
<code>ADC_ROUTINE_CHANNEL</code>	routine sequence
<code>ADC_INSERTED_CHANNEL</code>	inserted sequence
<b>Input parameter{in}</b>	
<b>length</b>	the length of the channel, routine channel 1-12, inserted channel 1-4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the length of ADC routine channel */
```

```
adc_channel_length_config(ADC_ROUTINE_CHANNEL, 4);
```

## adc\_routine\_channel\_config

The description of adc\_routine\_channel\_config is shown as below:

**Table 3-18. Function adc\_routine\_channel\_config**

<b>Function name</b>	adc_routine_channel_config
<b>Function prototype</b>	void adc_routine_channel_config(uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
<b>Function descriptions</b>	configure ADC routine channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rank</b>	the routine sequence rank, this parameter must be between 0 to 11
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the selected ADC channel
ADC_CHANNEL_x (x=0..14)	ADC Channelx (x=0..14)
<b>Input parameter{in}</b>	
<b>sample_time</b>	the sample time value
ADC_SAMPLETIME_1 POINT5	1.5cycles
ADC_SAMPLETIME_1 4POINT5	14.5cycles
ADC_SAMPLETIME_2 7POINT5	23.5cycles
ADC_SAMPLETIME_5 5POINT5	55.5cycles
ADC_SAMPLETIME_8 3POINT5	83.5cycles
ADC_SAMPLETIME_1 11POINT5	111.5cycles
ADC_SAMPLETIME_1 43POINT5	143.5cycles
ADC_SAMPLETIME_4 79POINT5	479.5cycles
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC routine channel */
```

```
adc_routine_channel_config(1, ADC_CHANNEL_0, ADC_SAMPLETIME_14POINT5);
```

### adc\_inserted\_channel\_config

The description of adc\_inserted\_channel\_config is shown as below:

**Table 3-19. Function adc\_inserted\_channel\_config**

<b>Function name</b>	adc_inserted_channel_config
<b>Function prototype</b>	void adc_inserted_channel_config(uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
<b>Function descriptions</b>	configure ADC inserted channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rank</b>	the inserted sequence rank, this parameter must be between 0 to 3
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the selected ADC channel
ADC_CHANNEL_x (x=0..14)	ADC Channelx (x=0..14)
<b>Input parameter{in}</b>	
<b>sample_time</b>	the sample time value
ADC_SAMPLETIME_1 POINT5	1.5cycles
ADC_SAMPLETIME_1 4POINT5	14.5cycles
ADC_SAMPLETIME_2 7POINT5	23.5cycles
ADC_SAMPLETIME_5 5POINT5	55.5cycles
ADC_SAMPLETIME_8 3POINT5	83.5cycles
ADC_SAMPLETIME_1 11POINT5	111.5cycles
ADC_SAMPLETIME_1 43POINT5	143.5cycles
ADC_SAMPLETIME_4 79POINT5	479.5cycles
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC inserted channel */
```

```
adc_inserted_channel_config(1, ADC_CHANNEL_0, ADC_SAMPLETIME_14POINT5);
```

### 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**

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

Example:

```
/* configure ADC inserted channel offset */
```

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

### adc\_external\_trigger\_config

The description of adc\_external\_trigger\_config is shown as below:

**Table 3-21. Function adc\_external\_trigger\_config**

<b>Function name</b>	adc_external_trigger_config
<b>Function prototype</b>	void adc_external_trigger_config(uint8_t adc_sequence, uint32_t trigger_mode);
<b>Function descriptions</b>	configure ADC external trigger
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence

<i>ADC_ROUTINE_CHANNEL</i>	routine sequence
<i>ADC_INSERTED_CHANNEL</i>	inserted sequence
<b>Input parameter{in}</b>	
<b>trigger_mode</b>	external trigger mode
<i>EXTERNAL_TRIGGER_DISABLE</i>	external trigger disable
<i>EXTERNAL_TRIGGER_RISING</i>	rising edge of external trigger
<i>EXTERNAL_TRIGGER_FALLING</i>	falling edge of external trigger
<i>EXTERNAL_TRIGGER_RISING_FALLING</i>	rising and falling edge of external trigger
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC inserted sequence external trigger */
```

```
adc_external_trigger_config(ADC_INSERTED_CHANNEL,
EXTERNAL_TRIGGER_RISING);
```

### adc\_external\_trigger\_source\_config

The description of `adc_external_trigger_source_config` is shown as below:

**Table 3-22. Function `adc_external_trigger_source_config`**

<b>Function name</b>	<code>adc_external_trigger_source_config</code>
<b>Function prototype</b>	<code>void adc_external_trigger_source_config(uint8_t adc_sequence, uint32_t external_trigger_source);</code>
<b>Function descriptions</b>	configure ADC external trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
<i>ADC_ROUTINE_CHANNEL</i>	routine sequence
<i>ADC_INSERTED_CHANNEL</i>	inserted sequence
<b>Input parameter{in}</b>	
<b>external_trigger_</b>	routine or inserted sequence trigger source



source	
ADC_EXTTRIG_ ROUTINE_T0_CH0	TIMER0 CH0 event select for routine channel
ADC_EXTTRIG_ ROUTINE_T0_CH1	TIMER0 CH1 event select for routine channel
ADC_EXTTRIG_ ROUTINE_T0_CH2	TIMER0 CH2 event select for routine channel
ADC_EXTTRIG_ ROUTINE_T1_CH1	TIMER1 CH1 event select for routine channel
ADC_EXTTRIG_ ROUTINE_T1_CH2	TIMER1 CH2 event select for routine channel
ADC_EXTTRIG_ ROUTINE_T1_CH3	TIMER1 CH3 event select for routine channel
ADC_EXTTRIG_ ROUTINE_T1_TRGO	TIMER1 TRGO event select for routine channel
ADC_EXTTRIG_ ROUTINE_T2_CH0	TIMER2 CH0 event select for routine channel
ADC_EXTTRIG_ ROUTINE_T2_TRGO	TIMER2 TRGO event select for routine channel
ADC_EXTTRIG_ ROUTINE_T3_CH3	TIMER3 CH3 event select for routine channel
ADC_EXTTRIG_ROUTINE_T4_CH0	TIMER4 CH0 event select for routine channel
ADC_EXTTRIG_ ROUTINE_T4_CH1	TIMER4 CH1 event select for routine channel
ADC_EXTTRIG_ ROUTINE_T4_CH2	TIMER4 CH2 event select for routine channel
ADC_EXTTRIG_ ROUTINE_EXTL_11	external interrupt line 11 for routine channel
ADC_EXTTRIG_ INSERTED_T0_CH3	TIMER0 CH3 event select for inserted channel
ADC_EXTTRIG_ INSERTED_T0_TRGO	TIMER0 TRGO event select for inserted channel
ADC_EXTTRIG_ INSERTED_T1_CH0	TIMER1 CH0 event select for inserted channel
ADC_EXTTRIG_ INSERTED_T1_TRGO	TIMER1 TRGO event select for inserted channel
ADC_EXTTRIG_ INSERTED_T2_CH1	TIMER2 CH1 event select for inserted channel
ADC_EXTTRIG_ INSERTED_T2_CH3	TIMER3 CH3 event select for inserted channel
ADC_EXTTRIG_ INSERTED_T3_CH0	TIMER3 CH0 event select for inserted channel

<i>ADC_EXTTRIG_INSERTED_T3_CH1</i>	TIMER3 CH1 event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_T3_CH2</i>	TIMER3 CH2 event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_T3_TRGO</i>	TIMER3 TRGO event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_T4_CH3</i>	TIMER4 CH3 event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_T4_TRGO</i>	TIMER4 TRGO event select for inserted channel
<i>ADC_EXTTRIG_INSERTED_EXTI_15</i>	external interrupt line 15 for inserted channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC routine channel external trigger source */
```

```
adc_external_trigger_source_config (ADC_ROUTINE_CHANNEL,  
ADC_EXTTRIG_ROUTINE_T0_CH0);
```

### adc\_software\_trigger\_enable

The description of `adc_software_trigger_enable` is shown as below:

**Table 3-23. Function `adc_software_trigger_enable`**

<b>Function name</b>	<code>adc_software_trigger_enable</code>
<b>Function prototype</b>	<code>void adc_software_trigger_enable(uint8_t adc_sequence);</code>
<b>Function descriptions</b>	enable ADC software trigger
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
<i>ADC_ROUTINE_CHANNEL</i>	routine sequence
<i>ADC_INSERTED_CHANNEL</i>	inserted sequence
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC routine sequence software trigger */
```

```
adc_software_trigger_enable(ADC_ROUTINE_CHANNEL);
```

## adc\_routine\_data\_read

The description of adc\_routine\_data\_read is shown as below:

**Table 3-24. Function adc\_routine\_data\_read**

<b>Function name</b>	adc_routine_data_read
<b>Function prototype</b>	uint16_t adc_routine_data_read(void);
<b>Function descriptions</b>	read ADC routine sequence data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	ADC conversion value (0~0xFFFF)

Example:

```
/* read ADC routine sequence data register */

uint16_t adc_value = 0;

adc_value = adc_routine_data_read();
```

## adc\_inserted\_data\_read

The description of adc\_inserted\_data\_read is shown as below:

**Table 3-25. Function adc\_inserted\_data\_read**

<b>Function name</b>	adc_inserted_data_read
<b>Function prototype</b>	uint16_t adc_inserted_data_read(void);
<b>Function descriptions</b>	read ADC inserted sequence data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	ADC conversion value (0~0xFFFF)

Example:

```
/* read ADC inserted sequence data register */
```

```
uint16_t adc_value = 0;
```

```
adc_value = adc_inserted_data_read();
```

## adc\_inserted\_data\_read

The description of adc\_inserted\_data\_read is shown as below:

**Table 3-26. Function adc\_inserted\_data\_read**

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

Example:

```
/* read ADC inserted sequence data register */
```

```
uint16_t adc_value = 0;
```

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

## adc\_latch\_data\_read

The description of adc\_latch\_data\_read is shown as below:

**Table 3-27. Function adc\_latch\_data\_read**

<b>Function name</b>	adc_latch_data_read
<b>Function prototype</b>	uint16_t adc_latch_data_read(uint8_t latch_data);
<b>Function descriptions</b>	read ADC latch data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>latch_data</b>	Latch data selection
ADC_LATCH_DATA_x	latch data x, x=0,1,2,3
<b>Output parameter{out}</b>	
-	-

Return value	
uint16_t	ADC conversion value (0 ~ 4095)

Example:

```
/* read ADC latch data register */
```

```
uint16_t adc_value = 0;
```

```
adc_value = adcLatchDataRead(ADC_LATCH_DATA_0);
```

### adcLatchDataSouceConfig

The description of adcLatchDataSouceConfig is shown as below:

**Table 3-28. Function adcLatchDataSouceConfig**

Function name	adcLatchDataSouceConfig
Function prototype	void adcLatchDataSouceConfig(uint32_t adc_periph, uint8_t latch_data, uint8_t adc_sequence, uint8_t rank);
Function descriptions	configure ADC latch data souce
Precondition	-
The called functions	-
Input parameter{in}	
latch_data	Latch data selection
ADC_LATCH_DATA_x	latch data x, x=0,1,2,3
Input parameter{in}	
adc_sequence	select the sequence
ADC_ROUTINE_CHANNEL	routine sequence
ADC_INSERTED_CHANNEL	inserted sequence
Input parameter{in}	
rank	the routine sequence rank, this parameter must be between 0 to 11 the inserted sequence rank, this parameter must be between 0 to 3
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC latch data source */
```

```
adcLatchDataSouceConfig(ADC_LATCH_DATA_0, ADC_ROUTINE_CHANNEL, 5);
```

### adcWatchdog0SingleChannelEnable

The description of adcWatchdog0SingleChannelEnable is shown as below:

**Table 3-29. Function adc\_watchdog0\_single\_channel\_enable**

<b>Function name</b>	adc_watchdog0_single_channel_enable
<b>Function prototype</b>	void adc_watchdog0_single_channel_enable(uint8_t adc_channel);
<b>Function descriptions</b>	configure ADC analog watchdog 0 single channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the selected ADC channel
<i>ADC_CHANNEL_x</i> (x=0..14)	ADC channelx(x=0..14)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC analog watchdog 0 single channel */
```

```
adc_watchdog0_single_channel_enable(ADC_CHANNEL_1);
```

## adc\_watchdog0\_sequence\_channel\_enable

The description of adc\_watchdog0\_sequence\_channel\_enable is shown as below:

**Table 3-30. Function adc\_watchdog0\_sequence\_channel\_enable**

<b>Function name</b>	adc_watchdog0_sequence_channel_enable
<b>Function prototype</b>	void adc_watchdog0_sequence_channel_enable(uint8_t adc_sequence);
<b>Function descriptions</b>	configure ADC analog watchdog 0 sequence channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	the sequence use analog watchdog 0
<i>ADC_ROUTINE_CHANNEL</i>	routine sequence
<i>ADC_INSERTED_CHANNEL</i>	inserted sequence
<i>ADC_ROUTINE_INSERTED_CHANNEL</i>	both routine and inserted sequence
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC analog watchdog 0 sequence */
```

```
adc_watchdog0_sequence_channel_enable(ADC_ROUTINE_CHANNEL);
```

### adc\_watchdog0\_disable

The description of adc\_watchdog0\_disable is shown as below:

**Table 3-31. Function adc\_watchdog0\_disable**

<b>Function name</b>	adc_watchdog0_disable
<b>Function prototype</b>	void adc_watchdog0_disable(void);
<b>Function descriptions</b>	disable ADC analog watchdog 0
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC analog watchdog 0 */
```

```
adc_watchdog0_disable();
```

### adc\_watchdog0\_threshold\_config

The description of adc\_watchdog0\_threshold\_config is shown as below:

**Table 3-32. Function adc\_watchdog0\_threshold\_config**

<b>Function name</b>	adc_watchdog0_threshold_config
<b>Function prototype</b>	void adc_watchdog0_threshold_config(uint16_t low_threshold , uint16_t high_threshold);
<b>Function descriptions</b>	configure ADC analog watchdog 0 threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>low_threshold</b>	analog watchdog0 low threshold, 0..2 <sup>20</sup> -1
<b>Input parameter{in}</b>	
<b>high_threshold</b>	analog watchdog0 high threshold, 0..2 <sup>20</sup> -1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC analog watchdog 0 threshold */
```

```
adc_watchdog0_threshold_config( 0x0400, 0x0A00);
```

### adc\_oversample\_mode\_config

The description of adc\_oversample\_mode\_config is shown as below:

**Table 3-33. Function adc\_oversample\_mode\_config**

<b>Function name</b>	adc_oversample_mode_config
<b>Function prototype</b>	void adc_oversample_mode_config(uint32_t mode, uint16_t shift, uint8_t ratio);
<b>Function descriptions</b>	configure ADC oversample mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	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
<b>Input parameter{in}</b>	
<b>shift</b>	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
<b>Input parameter{in}</b>	
<b>ratio</b>	ADC oversampling ratio



ADC_OVERSAMPLING_RATIO_MUL2	oversampling ratio multiple 2
ADC_OVERSAMPLING_RATIO_MUL4	oversampling ratio multiple 4
ADC_OVERSAMPLING_RATIO_MUL8	oversampling ratio multiple 8
ADC_OVERSAMPLING_RATIO_MUL16	oversampling ratio multiple 16
ADC_OVERSAMPLING_RATIO_MUL32	oversampling ratio multiple 32
ADC_OVERSAMPLING_RATIO_MUL64	oversampling ratio multiple 64
ADC_OVERSAMPLING_RATIO_MUL128	oversampling ratio multiple 128
ADC_OVERSAMPLING_RATIO_MUL256	oversampling ratio multiple 256
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_oversample_mode_config(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 3-34. Function adc\_oversample\_mode\_enable**

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

Example:

```
/* enable ADC oversample mode */
```

```
adc_oversample_mode_enable();
```

### adc\_oversample\_mode\_disable

The description of adc\_oversample\_mode\_disable is shown as below:

**Table 3-35. Function adc\_oversample\_mode\_disable**

<b>Function name</b>	adc_oversample_mode_disable
<b>Function prototype</b>	void adc_oversample_mode_disable(void);
<b>Function descriptions</b>	disable ADC oversample mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC oversample mode */
```

```
adc_oversample_mode_disable ();
```

### adc\_flag\_get

The description of adc\_flag\_get is shown as below:

**Table 3-36. Function adc\_flag\_get**

<b>Function name</b>	adc_flag_get
<b>Function prototype</b>	FlagStatus adc_flag_get(uint32_t flag);
<b>Function descriptions</b>	get the ADC flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	the adc flag bits
ADC_FLAG_WD0E	analog watchdog0 event flag
ADC_FLAG_EOC	end of sequence conversion flag
ADC_FLAG_EOC	end of routine sequence conversion flag
ADC_FLAG_EOIC	end of inserted sequence conversion flag
ADC_FLAG_STIC	start flag of inserted sequence
ADC_FLAG_STRC	start flag of routine sequence
ADC_FLAG_ROVF	routine data register overflow flag
ADC_FLAG_IOVF	inserted data register overflow flag

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

Example:

```
/* get the ADC analog watchdog0 flag bits */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_flag_get(ADC_FLAG_WD0E);
```

### adc\_flag\_clear

The description of adc\_flag\_clear is shown as below:

**Table 3-37. Function adc\_flag\_clear**

Function name	adc_flag_clear
Function prototype	void adc_flag_clear(uint32_t flag);
Function descriptions	clear the ADC flag bits
Precondition	-
The called functions	-
Input parameter{in}	
flag	the adc flag bits
ADC_FLAG_WD0E	analog watchdog0 event flag
ADC_FLAG_EOC	end of sequence conversion flag
ADC_FLAG_EOC	end of routine sequence conversion flag
ADC_FLAG_EOIC	end of inserted sequence conversion flag
ADC_FLAG_STIC	start flag of inserted sequence
ADC_FLAG_STRC	start flag of routine sequence
ADC_FLAG_ROVF	routine data register overflow flag
ADC_FLAG_IOVF	inserted data register overflow flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the ADC analog watchdog0 flag bits */
```

```
adc_flag_clear(ADC_FLAG_WD0E);
```

### adc\_interrupt\_enable

The description of adc\_interrupt\_enable is shown as below:

Table 3-38. Function `adc_interrupt_enable`

<b>Function name</b>	<code>adc_interrupt_enable</code>
<b>Function prototype</b>	<code>void adc_interrupt_enable(uint32_t interrupt);</code>
<b>Function descriptions</b>	enable ADC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the adc interrupt
<code>ADC_INT_WD0E</code>	analog watchdog0 interrupt
<code>ADC_INT_EOC</code>	end of sequence conversion interrupt
<code>ADC_INT_EORC</code>	end of routine sequence conversion interrupt
<code>ADC_INT_EOIC</code>	end of inserted sequence conversion interrupt
<code>ADC_INT_ROVF</code>	routine data register overflow interrupt
<code>ADC_INT_IOVF</code>	inserted data register overflow interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC analog watchdog 0 interrupt */
```

```
adc_interrupt_enable(ADC_INT_WD0E);
```

### **`adc_interrupt_disable`**

The description of `adc_interrupt_disable` is shown as below:

Table 3-39. Function `adc_interrupt_disable`

<b>Function name</b>	<code>adc_interrupt_disable</code>
<b>Function prototype</b>	<code>void adc_interrupt_disable(uint32_t interrupt);</code>
<b>Function descriptions</b>	disable ADC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the adc interrupt
<code>ADC_INT_WD0E</code>	analog watchdog0 interrupt
<code>ADC_INT_EOC</code>	end of sequence conversion interrupt
<code>ADC_INT_EORC</code>	end of routine sequence conversion interrupt
<code>ADC_INT_EOIC</code>	end of inserted sequence conversion interrupt
<code>ADC_INT_ROVF</code>	routine data register overflow interrupt
<code>ADC_INT_IOVF</code>	inserted data register overflow interrupt
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* disable ADC analog watchdog0 interrupt */
```

```
adc_interrupt_disable(ADC_INT_WD0E);
```

### adc\_interrupt\_flag\_get

The description of adc\_interrupt\_flag\_get is shown as below:

**Table 3-40. Function adc\_interrupt\_flag\_get**

Function name	adc_interrupt_flag_get
Function prototype	FlagStatus adc_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get the ADC interrupt bits
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	the adc interrupt bits
ADC_INT_FLAG_WD0E	analog watchdog 0 interrupt
ADC_INT_FLAG_EOC	end of sequence conversion interrupt
ADC_INT_FLAG_EORC	end of routine sequence conversion interrupt
ADC_INT_FLAG_EOIC	end of inserted sequence conversion interrupt
ADC_INT_FLAG_ROVF	routine data register overflow interrupt
ADC_INT_FLAG_IOVF	inserted data register overflow interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the ADC analog watchdog 0 interrupt bits*/
```

```
FlagStatus flag_value;
```

```
flag_value = adc_interrupt_flag_get(ADC_INT_FLAG_WD0E);
```

### adc\_interrupt\_flag\_clear

The description of adc\_interrupt\_flag\_clear is shown as below:

**Table 3-41. Function adc\_interrupt\_flag\_clear**

<b>Function name</b>	adc_interrupt_flag_clear
<b>Function prototype</b>	void adc_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear the ADC interrupt bits
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	the adc interrupt bits
ADC_INT_FLAG_WD0E	analog watchdog 0 interrupt
ADC_INT_FLAG_EOC	end of sequence conversion interrupt
ADC_INT_FLAG_EORC	end of routine sequence conversion interrupt
ADC_INT_FLAG_EOIC	end of inserted sequence conversion interrupt
ADC_INT_FLAG_ROVF	routine data register overflow interrupt
ADC_INT_FLAG_IOVF	inserted data register overflow interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the ADC analog watchdog 0 interrupt bits*/
adc_interrupt_flag_clear(ADC_INT_FLAG_WD0E);
```

### 3.3. CAU

The Cryptographic Acceleration Unit supports acceleration of DES, Triple-DES or AES (128,192, or 256) algorithms. The CAU registers are listed in chapter [3.3.1](#) the CAU firmware functions are introduced in chapter [3.3.2](#)

#### 3.3.1. Descriptions of Peripheral registers

CAU registers are listed in the table shown as below:

**Table 3-42. CAU Registers**

Registers	Descriptions
CAU_CTL	control register
CAU_STAT0	status register 0
CAU_DI	data input register

Registers	Descriptions
CAU_DO	data output register
CAU_DMAEN	DMA enable register
CAU_INTEN	interrupt enable register
CAU_STAT1	status register 1
CAU_INTF	interrupt flag register
CAU_KEY0H	key 0 high register
CAU_KEY0L	key 0 low register
CAU_KEY1H	key 1 high register
CAU_KEY1L	key 1 low register
CAU_KEY2H	key 2 high register
CAU_KEY2L	key 2 low register
CAU_KEY3H	key 3 high register
CAU_KEY3L	key 3 low register
CAU_IV0H	initial vector 0 high register
CAU_IV0L	initial vector 0 low register
CAU_IV1H	initial vector 1 high register
CAU_IV1L	initial vector 1 low register
CAU_GCMCCMCT XSx (x = 0..7)	GCM or CCM mode context switch register x
CAU_GCMCTXSx (x = 0..7)	GCM mode context switch register x

### 3.3.2. Descriptions of Peripheral functions

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

**Table 3-43. CAU firmware function**

Function name	Function description
cau_deinit	reset the CAU peripheral
cau_struct_para_init	initialize the CAU encrypt and decrypt parameter struct with the default values
cau_key_struct_para_init	initialize the key parameter struct with the default values
cau_iv_struct_para_init	initialize the vectors parameter struct with the default values
cau_context_struct_para_init	initialize the context parameter struct with the default values
cau_enable	enable the CAU peripheral
cau_disable	disable the CAU peripheral
cau_dma_enable	enable the CAU DMA interface
cau_dma_disable	disable the CAU DMA interface
cau_init	initialize the CAU peripheral
cau_aes_keysize_config	configure key size if use AES algorithm
cau_key_init	initialize the key parameters
cau_iv_init	initialize the vectors parameters

Function name	Function description
cau_phase_config	configure phase
cau_fifo_flush	flush the IN and OUT FIFOs
cau_enable_state_get	return whether CAU peripheral is enabled or disabled
cau_data_write	write data to the IN FIFO
cau_data_read	return the last data entered into the output FIFO
cau_context_save	save context before context switching
cau_context_restore	restore context after context switching
cau_aes_ecb	encrypt and decrypt using AES in ECB mode
cau_aes_cbc	encrypt and decrypt using AES in CBC mode
cau_aes_ctr	encrypt and decrypt using AES in CTR mode
cau_aes_cfb	encrypt and decrypt using AES in CFB mode
cau_aes_ofb	encrypt and decrypt using AES in OFB mode
cau_aes_gcm	encrypt and decrypt using AES in GCM mode
cau_aes_ccm	encrypt and decrypt using AES in CCM mode
cau_tdes_ecb	encrypt and decrypt using TDES in ECB mode
cau_tdes_cbc	encrypt and decrypt using TDES in CBC mode
cau_des_ecb	encrypt and decrypt using DES in ECB mode
cau_des_cbc	encrypt and decrypt using DES in CBC mode
cau_interrupt_enable	enable the CAU interrupts
cau_interrupt_disable	disable the CAU interrupts
cau_interrupt_flag_get	get the interrupt flag
cau_flag_get	get the CAU flag status

## Structure cau\_key\_parameter\_struct

**Table 3-44. Structure cau\_key\_parameter\_struct**

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

## Structure cau\_iv\_parameter\_struct

**Table 3-45. Structure cau\_iv\_parameter\_struct**

Member name	Function description
iv_0_high	init vector 0 high
iv_0_low	init vector 0 low



iv_1_high	init vector 1 high
iv_1_low	init vector 1 low

### Structure cau\_context\_parameter\_struct

**Table 3-46. Structure cau\_context\_parameter\_struct**

Member name	Function description
ctl_config	current configuration
iv_0_high	init vector 0 high
iv_0_low	init vector 0 low
iv_1_high	init vector 1 high
iv_1_low	init vector 1 low
key_0_high	key 0 high
key_0_low	key 0 low
key_1_high	key 1 high
key_1_low	key 1 low
key_2_high	key 2 high
key_2_low	key 2 low
key_3_high	key 3 high
key_3_low	key 3 low
gcmccmctxs[8]	GCM or CCM mode context switch
gcmctxs[8]	GCM mode context switch

### Structure cau\_parameter\_struct

**Table 3-47. Structure cau\_parameter\_struct**

Member name	Function description
alg_dir	algorithm directory
*key	key
key_size	key size in bytes
*iv	initialization vector
iv_size	iv size in bytes
*input	input data
in_length	input data length in bytes
*aad	additional authentication data
aad_size	aad size

### cau\_deinit

The description of cau\_deinit is shown as below:

**Table 3-48. Function cau\_deinit**

Function name	cau_deinit
Function prototype	void cau_deinit(void)

<b>Function descriptions</b>	reset the CAU peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset the CAU peripheral */
```

```
cau_deinit( );
```

## cau\_struct\_para\_init

The description of cau\_struct\_para\_init is shown as below:

**Table 3-49. Function cau\_struct\_para\_init**

<b>Function name</b>	cau_struct_para_init
<b>Function prototype</b>	void cau_struct_para_init(cau_parameter_struct *cau_parameter)
<b>Function descriptions</b>	initialize the CAU encrypt and decrypt parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>cau_parameter</b>	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-47. Structure cau_parameter_struct</a>
<b>Return value</b>	
-	-

Example:

```
cau_parameter_struct text;
```

```
/* initialize CAU encrypt and decrypt parameter struct */
```

```
cau_struct_para_init(&text);
```

## cau\_key\_struct\_para\_init

The description of cau\_key\_struct\_para\_init is shown as below:

Table 3-50. Function cau\_key\_struct\_para\_init

Function name	cau_key_struct_para_init
Function prototype	void cau_key_struct_para_init(cau_key_parameter_struct *key_initpara)
Function descriptions	initialize the key parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
key_initpara	the key parameters, refer to structure <a href="#">Table 3-44. Structure cau_key_parameter_struct</a>
Return value	
-	-

Example:

```
/* initialize the key parameter struct */
cau_key_parameter_struct key_initpara;
cau_key_struct_para_init(&key_initpara);
```

### cau\_iv\_struct\_para\_init

The description of cau\_iv\_struct\_para\_init is shown as below:

Table 3-51. Function cau\_iv\_struct\_para\_init

Function name	cau_iv_struct_para_init
Function prototype	void cau_iv_struct_para_init(cau_iv_parameter_struct *iv_initpara)
Function descriptions	initialize the vectors parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
iv_initpara	the vectors parameter, refer to structure <a href="#">Table 3-45. Structure cau_iv_parameter_struct</a>
Return value	
-	-

Example:

```
/* initialize the vectors parameter struct */
cau_iv_parameter_struct iv_initpara;
cau_iv_struct_para_init(&iv_initpara);
```

## cau\_context\_struct\_para\_init

The description of cau\_context\_struct\_para\_init is shown as below:

**Table 3-52. Function cau\_context\_struct\_para\_init**

<b>Function name</b>	cau_context_struct_para_init
<b>Function prototype</b>	void cau_context_struct_para_init(cau_context_parameter_struct *cau_context)
<b>Function descriptions</b>	initialize the context parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
cau_context	the context parameter, refer to structure <a href="#">Table 3-46. Structure cau_context_parameter_struct</a>
<b>Return value</b>	
-	-

Example:

```
/* initialize the context parameter struct */
cau_context_parameter_struct context_initpara;
cau_context_struct_para_init (&context_initpara);
```

## cau\_enable

The description of cau\_enable is shown as below:

**Table 3-53. Function cau\_enable**

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

Example:

```
/* enable the CAU peripheral */
```

cau\_enable();

## cau\_disable

The description of cau\_disable is shown as below:

**Table 3-54. Function cau\_disable**

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

Example:

```
/* disable the CAU peripheral */
```

```
cau_disable();
```

## cau\_dma\_enable

The description of cau\_dma\_enable is shown as below:

**Table 3-55. Function cau\_dma\_enable**

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

Example:

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

```
cau_dma_enable(CAU_DMA_INFIFO);
```

## cau\_dma\_disable

The description of cau\_dma\_disable is shown as below:

**Table 3-56. Function cau\_dma\_disable**

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

Example:

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

```
cau_dma_disable(CAU_DMA_INFIFO);
```

## cau\_init

The description of cau\_init is shown as below:

**Table 3-57. Function cau\_init**

<b>Function name</b>	cau_init
<b>Function prototype</b>	void cau_init(uint32_t alg_dir, uint32_t algo_mode, uint32_t swapping)
<b>Function descriptions</b>	initialize the CAU peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>alg_dir</b>	algorithm direction
CAU_ENCRYPT	encrypt
CAU_DECRYPT	decrypt
<b>Input parameter{in}</b>	
<b>algo_mode</b>	algorithm mode selection
CAU_MODE_TDES_ECB	TDES-ECB (3DES Electronic codebook)
CAU_MODE_TDES_CBC	TDES-CBC (3DES Cipher block chaining)

<i>BC</i>	
<i>CAU_MODE_DES_EC B</i>	DES-ECB (simple DES Electronic codebook)
<i>CAU_MODE_DES_CB C</i>	DES-CBC (simple DES Cipher block chaining)
<i>CAU_MODE_AES_EC B</i>	AES-ECB (AES Electronic codebook)
<i>CAU_MODE_AES_CB C</i>	AES-CBC (AES Cipher block chaining)
<i>CAU_MODE_AES_CT R</i>	AES-CTR (AES counter mode)
<i>CAU_MODE_AES_KE Y</i>	AES decryption key preparation mode
<i>CAU_MODE_AES_GC M</i>	AES-GCM (AES Galois/counter mode)
<i>CAU_MODE_AES_CC M</i>	AES-CCM (AES combined cipher machine mode)
<i>CAU_MODE_AES_CF B</i>	AES-CFB (cipher feedback mode)
<i>CAU_MODE_AES_OF B</i>	AES-OFB (output feedback mode)
<b>Input parameter{in}</b>	
<b>swapping</b>	data swapping selection
<i>CAU_SWAPPING_32BIT T</i>	no swapping
<i>CAU_SWAPPING_16BIT T</i>	half-word swapping
<i>CAU_SWAPPING_8BIT</i>	bytes swapping
<i>CAU_SWAPPING_1BIT</i>	bit swapping
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### cau\_aes\_keysize\_config

The description of cau\_aes\_keysize\_config is shown as below:

Table 3-58. Function cau\_aes\_keysize\_config

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

Example:

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

```
cau_aes_keysize_config(CAU_KEYSIZE_128BIT);
```

## cau\_key\_init

The description of cau\_key\_init is shown as below:

Table 3-59. Function cau\_key\_init

Function name	cau_key_init
Function prototype	void cau_key_init(cau_key_parameter_struct* key_initpara);
Function descriptions	initialize the key parameters
Precondition	-
The called functions	-
Input parameter{in}	
key_initpara	the key parameters, refer to structure <a href="#">Table 3-44. Structure cau_key_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
cau_key_parameter_struct key_initpara;
```

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



## cau\_iv\_init

The description of cau\_iv\_init is shown as below:

**Table 3-60. Function cau\_iv\_init**

<b>Function name</b>	cau_iv_init
<b>Function prototype</b>	void cau_iv_init(cau_iv_parameter_struct* iv_initpara);
<b>Function descriptions</b>	initialize the vectors parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>iv_initpara</b>	the vectors parameters, refer to structure <a href="#">Table 3-45. Structure cau_iv_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the vectors parameters */
cau_iv_parameter_struct iv_initpara;
cau_iv_init(&iv_initpara);
```

## cau\_phase\_config

The description of cau\_phase\_config is shown as below:

**Table 3-61. Function cau\_phase\_config**

<b>Function name</b>	cau_phase_config
<b>Function prototype</b>	void cau_phase_config(uint32_t phase)
<b>Function descriptions</b>	configure phase
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>phase</b>	gcm or ccm phase
<i>CAU_PREPARE_PHASE</i>	prepare phase
<i>CAU_AAD_PHASE</i>	AAD phase
<i>CAU_ENCRYPT_DECRYPT_PHASE</i>	encryption/decryption phase
<i>CAU_TAG_PHASE</i>	tag phase
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* select prepare phase */
cau_phase_config(CAU_PREPARE_PHASE);
```

### cau\_fifo\_flush

The description of cau\_fifo\_flush is shown as below:

**Table 3-62. Function cau\_fifo\_flush**

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

Example:

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

### cau\_enable\_state\_get

The description of cau\_enable\_state\_get is shown as below:

**Table 3-63. Function cau\_enable\_state\_get**

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

Example:

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

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

## cau\_data\_write

The description of cau\_data\_write is shown as below:

**Table 3-64. Function cau\_data\_write**

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

Example:

```
/* write data to the IN FIFO */
```

```
cau_data_write(0x10);
```

## cau\_data\_read

The description of cau\_data\_read is shown as below:

**Table 3-65. Function cau\_data\_read**

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

Example:

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

```
uint32_t data;
```

```
data = cau_data_read();
```

### cau\_context\_save

The description of cau\_context\_save is shown as below:

**Table 3-66. Function cau\_context\_save**

<b>Function name</b>	cau_context_save
<b>Function prototype</b>	void cau_context_save(cau_context_parameter_struct *cau_context, cau_key_parameter_struct* key_initpara)
<b>Function descriptions</b>	save context before context switching
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>key_initpara</b>	the key parameters, refer to structure <a href="#">Table 3-44. Structure cau_key_parameter_struct</a>
<b>Output parameter{out}</b>	
<b>cau_context</b>	the context, refer to structure <a href="#">Table 3-46. Structure cau_context_parameter_struct</a>
<b>Return value</b>	
-	-

Example:

```
cau_context_parameter_struct context;

cau_key_parameter_struct key;

cau_parameter_struct cau_parameter;

uint32_t keyaddr;

.....

keyaddr = (uint32_t)(cau_parameter->key);

cau_key_struct_para_init(&key);

key.key_1_high = __REV(*(uint32_t*)(keyaddr));

keyaddr += 4U;

key.key_1_low= __REV(*(uint32_t*)(keyaddr));

/* save context before context switching */

cau_context_save(&context, &key);
```

## cau\_context\_restore

The description of cau\_context\_restore is shown as below:

**Table 3-67. Function cau\_context\_restore**

<b>Function name</b>	cau_context_restore
<b>Function prototype</b>	void cau_context_restore(cau_context_parameter_struct *cau_context);
<b>Function descriptions</b>	restore context after context switching
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cau_context</b>	the context, refer to structure <a href="#">Table 3-46. Structure cau_context_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
cau_context_parameter_struct context;

.....

cau_context_save(&context, &key);

.....

/* restore context after context switching */

cau_context_restore (&context);
```

## cau\_aes\_ecb

The description of cau\_aes\_ecb is shown as below:

**Table 3-68. Function cau\_aes\_ecb**

<b>Function name</b>	cau_aes_ecb
<b>Function prototype</b>	ErrStatus cau_aes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);
<b>Function descriptions</b>	encrypt and decrypt using AES in ECB mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cau_parameter</b>	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-47. Structure cau_parameter_struct</a>
<b>Output parameter{out}</b>	
<b>output</b>	pointer to the returned buffer

Return value	
ErrStatus	SUCCESS or ERROR

Example:

```

cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.input      = plaintext;

text.in_length  = TEXT_SIZE;

/* encryption in ECB mode */

status = cau_aes_ecb(&text, encrypt_result);

```

### cau\_aes\_cbc

The description of cau\_aes\_cbc is shown as below:

**Table 3-69. Function cau\_aes\_cbc**

<b>Function name</b>	cau_aes_cbc
<b>Function prototype</b>	ErrStatus cau_aes_cbc(cau_parameter_struct *cau_parameter, uint8_t *output)
<b>Function descriptions</b>	encrypt and decrypt using AES in CBC mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-47. Structure cau_parameter_struct</a>
<b>Output parameter{out}</b>	
output	pointer to the returned buffer
<b>Return value</b>	
ErrStatus	SUCCESS or ERROR

Example:

```
cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.iv         = vectors;

text.input      = plaintext;

text.in_length  = TEXT_SIZE;

/* encryption in CBC mode */

status = cau_aes_cbc(&text, encrypt_result);
```

### cau\_aes\_ctr

The description of cau\_aes\_ctr is shown as below:

**Table 3-70. Function cau\_aes\_ctr**

<b>Function name</b>	cau_aes_ctr
<b>Function prototype</b>	ErrStatus cau_aes_ctr(cau_parameter_struct *cau_parameter, uint8_t *output)
<b>Function descriptions</b>	encrypt and decrypt using AES in CTR mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cau_parameter</b>	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-47. Structure cau_parameter_struct</a>
<b>Output parameter{out}</b>	
<b>output</b>	pointer to the returned buffer
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```

cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.iv         = vectors;

text.input      = plaintext;

text.in_length  = TEXT_SIZE;

/* encryption in CTR mode */

status = cau_aes_ctr(&text, encrypt_result);

```

### cau\_aes\_cfb

The description of cau\_aes\_cfb is shown as below:

**Table 3-71. Function cau\_aes\_cfb**

<b>Function name</b>	cau_aes_cfb
<b>Function prototype</b>	ErrStatus cau_aes_cfb(cau_parameter_struct *cau_parameter, uint8_t *output)
<b>Function descriptions</b>	encrypt and decrypt using AES in CFB mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cau_parameter</b>	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-47. Structure cau_parameter_struct</a>
<b>Output parameter{out}</b>	
<b>output</b>	pointer to the returned buffer
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
cau_parameter_struct cau_cfb_parameter;
```



```
uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&cau_cfb_parameter);

/* encryption in CFB mode */

cau_cfb_parameter.alg_dir    = CAU_ENCRYPT;

cau_cfb_parameter.key        = (uint8_t *)key_128;

cau_cfb_parameter.key_size   = KEY_SIZE;

cau_cfb_parameter.iv         = (uint8_t *)vectors;

cau_cfb_parameter.iv_size    = IV_SIZE;

cau_cfb_parameter.input       = (uint8_t *)plaintext;

cau_cfb_parameter.in_length = PLAINTEXT_SIZE;

status = cau_aes_cfb(&cau_cfb_parameter, encrypt_result);
```

## cau\_aes\_ofb

The description of cau\_aes\_ofb is shown as below:

**Table 3-72. Function cau\_aes\_ofb**

<b>Function name</b>	cau_aes_ofb
<b>Function prototype</b>	ErrStatus cau_aes_ofb(cau_parameter_struct *cau_parameter, uint8_t *output)
<b>Function descriptions</b>	encrypt and decrypt using AES in OFB mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cau_parameter</b>	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-47. Structure cau_parameter_struct</a>
<b>Output parameter{out}</b>	
<b>output</b>	pointer to the returned buffer
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
cau_parameter_struct cau_ofb_parameter;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;
```

.....

```
cau_struct_para_init(&cau_ofb_parameter);

/* encryption in OFB mode */

cau_ofb_parameter.alg_dir    = CAU_ENCRYPT;
cau_ofb_parameter.key        = (uint8_t *)key_128;
cau_ofb_parameter.key_size   = KEY_SIZE;
cau_ofb_parameter.iv         = (uint8_t *)vectors;
cau_ofb_parameter.iv_size    = IV_SIZE;
cau_ofb_parameter.input      = (uint8_t *)plaintext;
cau_ofb_parameter.in_length  = PLAINTEXT_SIZE;

status = cau_aes_ofb(&cau_ofb_parameter, encrypt_result);
```

## cau\_aes\_gcm

The description of cau\_aes\_gcm is shown as below:

**Table 3-73. Function cau\_aes\_gcm**

Function name	cau_aes_gcm
Function prototype	ErrStatus cau_aes_gcm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t *tag)
Function descriptions	encrypt and decrypt using AES in GCM mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-47. Structure cau_parameter_struct</a>
Output parameter{out}	
output	pointer to the returned buffer
Output parameter{out}	
tag	pointer to the returned tag buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
cau_parameter_struct cau_gcm_parameter;

uint8_t encrypt_result[TEXT_SIZE];

uint8_t gcm_tag[GCM_TAG_SIZE];

ErrStatus status;
```

.....

```
cau_struct_para_init(&cau_gcm_parameter);

/* encryption in GCM mode */

cau_gcm_parameter.alg_dir    = CAU_ENCRYPT;

cau_gcm_parameter.key        = (uint8_t *)key_128;

cau_gcm_parameter.key_size   = KEY_SIZE;

cau_gcm_parameter.iv         = (uint8_t *)vectors;

cau_gcm_parameter.iv_size    = IV_SIZE;

cau_gcm_parameter.input      = (uint8_t *)plaintext;

cau_gcm_parameter.in_length   = PLAINTEXT_SIZE;

cau_gcm_parameter.aad        = (uint8_t *)aadmessage;

cau_gcm_parameter.aad_size = AAD_SIZE;

status = cau_aes_gcm(&cau_gcm_parameter, encrypt_result, gcm_tag);
```

### cau\_aes\_ccm

The description of cau\_aes\_ccm is shown as below:

**Table 3-74. Function cau\_aes\_ccm**

Function name	cau_aes_ccm
Function prototype	ErrStatus cau_aes_ccm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t tag[], uint32_t tag_size, uint8_t aad_buf[])
Function descriptions	encrypt and decrypt using AES in CCM mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-47. Structure cau_parameter_struct</a>
Input parameter{in}	
tag_size	tag size (in bytes)
Output parameter{out}	
output	pointer to the returned buffer
Output parameter{out}	
tag	pointer to the returned tag buffer
Output parameter{out}	
aad_buf	pointer to the user buffer used when formatting aad block
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
cau_parameter_struct cau_ccm_parameter;

uint8_t encrypt_result[TEXT_SIZE];

uint8_t ccm_tag[CCM_TAG_SIZE];

uint8_t aad_buf[AAD_SIZE + 21];

ErrStatus status;

.....

cau_struct_para_init(&cau_ccm_parameter);

/* encryption in CCM mode */

cau_ccm_parameter.alg_dir    = CAU_ENCRYPT;

cau_ccm_parameter.key        = (uint8_t *)ccm_key_128;

cau_ccm_parameter.key_size   = KEY_SIZE;

cau_ccm_parameter.iv         = (uint8_t *)ccm_vectors;

cau_ccm_parameter.iv_size    = CCM_IV_SIZE;

cau_ccm_parameter.input      = (uint8_t *)plaintext;

cau_ccm_parameter.in_length  = PLAINTEXT_SIZE;

cau_ccm_parameter.aad        = (uint8_t *)aadmessage;

cau_ccm_parameter.aad_size   = AAD_SIZE;

status = cau_aes_ccm(&cau_ccm_parameter, encrypt_result, ccm_tag, CCM_TAG_SIZE,
(uint8_t *)aad_buf);
```

## cau\_tdes\_ecb

The description of cau\_tdes\_ecb is shown as below:

**Table 3-75. Function cau\_tdes\_ecb**

<b>Function name</b>	cau_tdes_ecb
<b>Function prototype</b>	ErrStatus cau_tdes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output)
<b>Function descriptions</b>	encrypt and decrypt using TDES in ECB mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-47. Structure cau_parameter_struct</a>

Output parameter{out}	
<b>output</b>	pointer to the returned buffer
Return value	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

text.alg_dir   = CAU_ENCRYPT;

text.key       = tdes_key;

text.input     = plaintext;

text.in_length = DATA_SIZE;

/* encryption in ECB mode */

status = cau_tdes_ecb(&text, encrypt_result);
```

### cau\_tdes\_cbc

The description of cau\_tdes\_cbc is shown as below:

**Table 3-76. Function cau\_tdes\_cbc**

<b>Function name</b>	cau_tdes_cbc
<b>Function prototype</b>	ErrStatus cau_tdes_cbc(cau_parameter_struct *cau_parameter, uint8_t *output)
<b>Function descriptions</b>	encrypt and decrypt using TDES in CBC mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>cau_parameter</b>	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-47. Structure cau_parameter_struct</a>
Output parameter{out}	
<b>output</b>	pointer to the returned buffer
Return value	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
cau_parameter_struct text;
```

```
uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

text.alg_dir    = CAU_ENCRYPT;

text.key        = tdes_key;

text.iv         = vectors;

text.input      = plaintext;

text.in_length  = DATA_SIZE;

/* encryption in CBC mode */

status = cau_tdes_cbc(&text, encrypt_result);
```

## cau\_des\_ecb

The description of cau\_des\_ecb is shown as below:

**Table 3-77. Function cau\_des\_ecb**

<b>Function name</b>	cau_des_ecb
<b>Function prototype</b>	ErrStatus cau_des_ecb(cau_parameter_struct *cau_parameter, uint8_t *output)
<b>Function descriptions</b>	encrypt and decrypt using DES in ECB mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cau_parameter</b>	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-47. Structure cau_parameter_struct</a>
<b>Output parameter{out}</b>	
<b>output</b>	pointer to the returned buffer
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);
```

```

text.alg_dir   = CAU_ENCRYPT;

text.key       = des_key;

text.input     = plaintext;

text.in_length = DATA_SIZE;

/* encryption in ECB mode */

status = cau_des_ecb(&text, encrypt_result);

```

### cau\_des\_cbc

The description of cau\_des\_cbc is shown as below:

**Table 3-78. Function cau\_des\_cbc**

<b>Function name</b>	cau_des_cbc
<b>Function prototype</b>	ErrStatus cau_des_cbc(cau_parameter_struct *cau_parameter, uint8_t *output)
<b>Function descriptions</b>	encrypt and decrypt using DES in CBC mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cau_parameter</b>	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-47. Structure cau_parameter_struct</a>
<b>Output parameter{out}</b>	
<b>output</b>	pointer to the returned buffer
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```

cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

text.alg_dir   = CAU_ENCRYPT;

text.key       = des_key;

text.iv        = vectors;

text.input     = plaintext;

text.in_length = DATA_SIZE;

```

```
/* encryption in CBC mode */
```

```
status = cau_des_cbc(&text, encrypt_result);
```

## cau\_interrupt\_enable

The description of cau\_interrupt\_enable is shown as below:

**Table 3-79. Function cau\_interrupt\_enable**

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

Example:

```
/* enable cau interrupt */
```

```
cau_interrupt_enable (CAU_INT_INFIFO);
```

## cau\_interrupt\_disable

The description of cau\_interrupt\_disable is shown as below:

**Table 3-80. Function cau\_interrupt\_disable**

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



Example:

```
/* disable cau interrupt */

cau_interrupt_disable(CAU_INT_INFIFO);
```

## cau\_interrupt\_flag\_get

The description of cau\_interrupt\_flag\_get is shown as below:

**Table 3-81. Function cau\_interrupt\_flag\_get**

<b>Function name</b>	cau_interrupt_flag_get
<b>Function prototype</b>	FlagStatus cau_interrupt_flag_get(uint32_t interrupt)
<b>Function descriptions</b>	get the interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	CAU interrupt flag
CAU_INT_FLAG_INFIFO	input FIFO interrupt
CAU_INT_FLAG_OUTFIFO	output FIFO interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

FlagStatus status = cau_interrupt_flag_get(CAU_INT_FLAG_INFIFO);
```

## cau\_flag\_get

The description of cau\_flag\_get is shown as below:

**Table 3-82. Function cau\_flag\_get**

<b>Function name</b>	cau_flag_get
<b>Function prototype</b>	FlagStatus cau_flag_get(uint32_t flag)
<b>Function descriptions</b>	get the CAU flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	CAU flag status
CAU_FLAG_INFIFO_EMPTY	input FIFO empty

CAU_FLAG_INFIFO_NO_FULL:	input FIFO is not full
CAU_FLAG_OUTFIFO_NO_EMPTY	output FIFO not empty
CAU_FLAG_OUTFIFO_FULL	output FIFO is full
CAU_FLAG_BUSY	the CAU core is busy
CAU_FLAG_INFIFO	input FIFO flag status
CAU_FLAG_OUTFIFO	output FIFO flag status
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

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

```
FlagStatus status;
```

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

## 3.4. CRC

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

### 3.4.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

**Table 3-83. CRC Registers**

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

### 3.4.2. Descriptions of Peripheral functions

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

**Table 3-84. CRC firmware function**

Function name	Function description
crc_deinit	deinit CRC calculation unit

Function name	Function description
crc_data_register_reset	reset the data register (CRC_DATA) to 0xFFFFFFFF
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_single_data_calculate	CRC calculate single 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 3-85. Function crc\_deinit**

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

Example:

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

## crc\_data\_register\_reset

The description of crc\_data\_register\_reset is shown as below:

**Table 3-86. Function crc\_data\_register\_reset**

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

-	-
---	---

Example:

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

```
crc_data_register_reset();
```

### crc\_data\_register\_read

The description of crc\_data\_register\_read is shown as below:

**Table 3-87. Function crc\_data\_register\_read**

<b>Function name</b>	crc_data_register_read
<b>Function prototype</b>	uint32_t crc_data_register_read(void);
<b>Function descriptions</b>	read the data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-88. Function crc\_free\_data\_register\_read**

<b>Function name</b>	crc_free_data_register_read
<b>Function prototype</b>	uint8_t crc_free_data_register_read(void);
<b>Function descriptions</b>	read the free data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

<b>uint8_t</b>	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 3-89. Function `crc_free_data_register_write`**

<b>Function name</b>	<code>crc_free_data_register_write</code>
<b>Function prototype</b>	<code>void crc_free_data_register_write(uint8_t free_data);</code>
<b>Function descriptions</b>	write the free data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>free_data</b>	specified 8-bit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

crc_free_data_register_write(0x11);
```

### **crc\_single\_data\_calculate**

The description of `crc_single_data_calculate` is shown as below:

**Table 3-90. Function `crc_single_data_calculate`**

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

<b>uint32_t</b>	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 3-91. Function `crc_block_data_calculate`**

<b>Function name</b>	<code>crc_block_data_calculate</code>
<b>Function prototype</b>	<code>uint32_t crc_block_data_calculate(uint32_t *array, uint32_t size);</code>
<b>Function descriptions</b>	CRC calculate a 32-bit data array
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>array</b>	pointer to the 32-bit input data array
<b>Input parameter{in}</b>	
<b>size</b>	size of the array
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	CRC calculate value (0-0xFFFFFFFF)

Example:

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

#define BUFFER_SIZE    6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

0x00001111, 0x00002222, 0x00003333, 0x00004444, 0x00005555, 0x00006666};

rcu_periph_clock_enable(RCU_CRC);

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

## 3.5. CTC

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

### 3.5.1. Descriptions of Peripheral registers

CTC registers are listed in the table shown as below:

**Table 3-92. CTC Registers**

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

### 3.5.2. Descriptions of Peripheral functions

CTC registers are listed in the table shown as below:

**Table 3-93. CTC firmware function**

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

Function name	Function description
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 3-94. 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 3-95. 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 3-96. Function ctc\_counter\_disable**

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

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 3-97. Function ctc\_irc48m\_trim\_value\_config**

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

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 3-98. Function ctc\_software\_refsource\_pulse\_generate**

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

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 3-99. Function ctc\_hardware\_trim\_mode\_config**

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

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 3-100. 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_REFSOURCE_POLARITY_FALLING	reference signal source polarity is falling edge
CTC_REFSOURCE_POLARITY_RISING	reference signal source polarity is rising edge
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set reference source polarity */
```

```
ctc_refsource_polarity_config (CTC_REFSOURCE_POLARITY_RISING);
```

### ctc\_refsource\_signal\_select

The description of ctc\_refsource\_signal\_select is shown as below:

**Table 3-101. Function ctc\_refsource\_signal\_select**

Function name	ctc_refsource_signal_select
Function prototype	void ctc_refsource_signal_select(uint32_t refs);
Function descriptions	select reference signal source
Precondition	-
The called functions	-
Input parameter{in}	
refs	reference signal source

<i>CTC_REFSOURCE_GPIO</i>	GPIO is selected
<i>CTC_REFSOURCE_LXTAL</i>	LXTAL is selected
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reference signal selection */
```

```
ctc_refsource_signal_select (CTC_REFSOURCE_LXTAL);
```

### ctc\_refsource\_prescaler\_config

The description of ctc\_refsource\_prescaler\_config is shown as below:

**Table 3-102. Function ctc\_refsource\_prescaler\_config**

<b>Function name</b>	ctc_refsource_prescaler_config
<b>Function prototype</b>	void ctc_refsource_prescaler_config(uint32_t prescaler);
<b>Function descriptions</b>	configure reference signal source prescaler
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>prescaler</b>	Prescaler factor
<i>CTC_REFSOURCE_PSC_OFF</i>	reference signal not divided
<i>CTC_REFSOURCE_PSC_DIV2</i>	reference signal divided by 2
<i>CTC_REFSOURCE_PSC_DIV4</i>	reference signal divided by 4
<i>CTC_REFSOURCE_PSC_DIV8</i>	reference signal divided by 8
<i>CTC_REFSOURCE_PSC_DIV16</i>	reference signal divided by 16
<i>CTC_REFSOURCE_PSC_DIV32</i>	reference signal divided by 32
<i>CTC_REFSOURCE_PSC_DIV64</i>	reference signal divided by 64
<i>CTC_REFSOURCE_PSC_DIV128</i>	reference signal divided by 128
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure reference signal source prescaler */
```

```
ctc_refsource_prescaler_config(CTC_REFSOURCE_PSC_DIV2);
```

### ctc\_clock\_limit\_value\_config

The description of ctc\_clock\_limit\_value\_config is shown as below:

**Table 3-103. 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 3-104. 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 3-105. Function ctc\_counter\_capture\_value\_read**

<b>Function name</b>	ctc_counter_capture_value_read
<b>Function prototype</b>	uint16_t ctc_counter_capture_value_read(void);
<b>Function descriptions</b>	read CTC counter capture value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-106. Function ctc\_counter\_direction\_read**

<b>Function name</b>	ctc_counter_direction_read
<b>Function prototype</b>	FlagStatus ctc_counter_direction_read(void);
<b>Function descriptions</b>	read CTC trim counter direction
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-107. Function ctc\_counter\_reload\_value\_read**

<b>Function name</b>	ctc_counter_reload_value_read
<b>Function prototype</b>	uint16_t ctc_counter_reload_value_read(void);
<b>Function descriptions</b>	read CTC counter reload value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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\_irc48m\_trim\_value\_read

The description of ctc\_irc48m\_trim\_value\_read is shown as below:

**Table 3-108. Function ctc\_irc48m\_trim\_value\_read**

<b>Function name</b>	ctc_irc48m_trim_value_read
<b>Function prototype</b>	uint8_t ctc_irc48m_trim_value_read(void);
<b>Function descriptions</b>	read the IRC48M trim value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

<b>uint8_t</b>	the 8-bit IRC48M trim value (0-63)
----------------	------------------------------------

Example:

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

```
uint8_t ctc_trim_value = 0;
```

```
ctc_trim_value = ctc_irc48m_trim_value_read ();
```

### ctc\_flag\_get

The description of ctc\_flag\_get is shown as below:

**Table 3-109. Function ctc\_flag\_get**

<b>Function name</b>	ctc_flag_get
<b>Function prototype</b>	FlagStatus ctc_flag_get(uint32_t flag);
<b>Function descriptions</b>	get CTC status flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	CTC status flag
CTC_FLAG_CKOK	clock trim OK flag
CTC_FLAG_CKWARN	clock trim warning flag
CTC_FLAG_ERR	error flag
CTC_FLAG_EREFP	expect reference flag
CTC_FLAG_CKERR	clock trim error bit
CTC_FLAG_REFMISS	reference sync pulse miss flag
CTC_FLAG_TRIMERR	trim value error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-110. Function ctc\_flag\_clear**

<b>Function name</b>	ctc_flag_clear
<b>Function prototype</b>	void ctc_flag_clear (uint32_t flag);
<b>Function descriptions</b>	clear CTC status flag



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	CTC status flag
<i>CTC_FLAG_CKOK</i>	clock trim OK flag
<i>CTC_FLAG_CKWARN</i>	clock trim warning flag
<i>CTC_FLAG_ERR</i>	error flag
<i>CTC_FLAG_EREFP</i>	expect reference flag
<i>CTC_FLAG_CKERR</i>	clock trim error bit
<i>CTC_FLAG_REFMISS</i>	reference sync pulse miss flag
<i>CTC_FLAG_TRIMERR</i>	trim value error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-111. Function ctc\_interrupt\_enable**

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

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 3-112. Function ctc\_interrupt\_disable**

<b>Function name</b>	ctc_interrupt_disable
<b>Function prototype</b>	void ctc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable the CTC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-113. Function ctc\_interrupt\_flag\_get**

<b>Function name</b>	ctc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus ctc_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	get CTC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	CTC interrupt flag
CTC_INT_FLAG_CKOK	clock trim OK interrupt
CTC_INT_FLAG_CKWARN	clock trim warning interrupt
CTC_INT_FLAG_ERR	error interrupt

<i>CTC_INT_FLAG_EREFR</i>	expect reference interrupt
<i>CTC_INT_FLAG_CKERR</i>	clock trim error bit interrupt
<i>CTC_INT_FLAG_REFMISS</i>	reference sync pulse miss interrupt
<i>CTC_INT_FLAG_TRIMERR</i>	trim value error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-114. Function ctc\_interrupt\_flag\_clear**

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

Example:

```
/* clear CTC interrupt flag status */

ctc_interrupt_flag_clear (CTC_INT_FLAG_CKOK);
```

## 3.6. DBG

The DBG hold unit helps debugger to debug power saving mode. The DBG registers are listed in chapter [3.6.1](#), the DBG firmware functions are introduced in chapter [3.6.2](#).

### 3.6.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

**Table 3-115. DBG Registers**

Registers	Descriptions
DBG_ID	DBG ID code register
DBG_CTL0	DBG control register0
DBG_CTL1	DBG control register1
DBG_CTL2	DBG control register2

### 3.6.2. Descriptions of Peripheral functions

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

**Table 3-116. 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 3-117. Enum dbg\_periph\_enum**

Member name	Function description
DBG_TIMER1_HOLD	hold TIMER1 counter when core is halted

Member name	Function description
DBG_TIMER2_HOLD	hold TIMER2 counter when core is halted
DBG_TIMER3_HOLD	hold TIMER3 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_RTC_HOLD	hold RTC counter when core is halted
DBG_WWDGT_HOLD	hold WWDGT counter when core is halted
DBG_FWDGT_HOLD	hold FWDGT counter when core is halted
DBG_I2C0_HOLD	hold I2C0 smbus timeout when core is halted
DBG_I2C1_HOLD	hold I2C1 smbus timeout when core is halted
DBG_TIMER0_HOLD	hold TIMER0 counter when core is halted
DBG_TIMER15_HOLD	hold TIMER15 counter when core is halted
DBG_TIMER16_HOLD	hold TIMER16 counter when core is halted

## dbg\_deinit

The description of dbg\_deinit is shown as below:

**Table 3-118. 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 3-119. 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	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-120. Function dbg\_low\_power\_enable**

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

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 3-121. Function `dbg_low_power_disable`**

<b>Function name</b>	<code>dbg_low_power_disable</code>
<b>Function prototype</b>	<code>void dbg_low_power_disable(uint32_t dbg_low_power);</code>
<b>Function descriptions</b>	disable low power behavior when the mcu is in debug mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b><code>dbg_low_power</code></b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-122. Function `dbg_periph_enable`**

<b>Function name</b>	<code>dbg_periph_enable</code>
<b>Function prototype</b>	<code>void dbg_periph_enable(dbg_periph_enum dbg_periph);</code>
<b>Function descriptions</b>	enable peripheral behavior when the mcu is in debug mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b><code>dbg_periph</code></b>	peripheral refer to <a href="#">Table 3-117. Enum <code>dbg_periph_enum</code></a>
<code>DBG_FWDGT_HOLD</code>	debug FWDGT kept when core is halted
<code>DBG_WWDGT_HOLD</code>	debug WWDGT kept when core is halted
<code>DBG_I2Cx_HOLD</code>	x=0,1 hold I2Cx smbus when core is halted
<code>DBG_TIMERx_HOLD</code>	x=0,1,2,3,4,5,15,16, hold TIMERx counter when core is halted
<code>DBG_RTC_HOLD</code>	hold RTC counter when core is halted
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

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 3-123. 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 <a href="#">Table 3-117. Enum dbg_periph_enum</a>
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted
DBG_I2Cx_HOLD	x=0,1 hold I2Cx smbus when core is halted
DBG_TIMERx_HOLD	x=0,1,2,3,4,5,15,16, hold TIMERx counter when core is halted
DBG_RTC_HOLD	hold RTC counter when core is halted
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 3-124. 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 3-125. 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();
```

## 3.7. DCI

The DCI is a parallel interface able to capture video or picture from a camera via Digital Camera Interface. The DCI registers are listed in chapter [3.7.1](#). the DCI firmware functions are introduced in chapter [3.7.2](#).

### 3.7.1. Descriptions of Peripheral registers

DCI registers are listed in the table shown as below:

**Table 3-126. DCI Registers**

Registers	Descriptions
DCI_CTL	DCI control register

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

## 3.7.2. Descriptions of Peripheral functions

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

**Table 3-127. DCI firmware function**

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

## Structure dci\_parameter\_struct

**Table 3-128. Structure dci\_parameter\_struct**

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

## dci\_deinit

The description of dci\_deinit is shown as below:

**Table 3-129. Function dci\_deinit**

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

Example:

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

```
dci_deinit();
```

## dci\_struct\_para\_init

The description of dci\_struct\_para\_init is shown as below:

Table 3-130. Function dci\_struct\_para\_init

<b>Function name</b>	dci_struct_para_init
<b>Function prototype</b>	void dci_struct_para_init(dci_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize dci_parameter_struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dci_struct</b>	address of DCI parameter initialization struct
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* initialize DCI struct parameter */

dci_parameter_struct  dci_struct;

dci_struct_para_init(&dci_struct);

```

## dci\_init

The description of dci\_init is shown as below:

Table 3-131. Function dci\_init

<b>Function name</b>	dci_init
<b>Function prototype</b>	void dci_init(dci_parameter_struct* dci_struct);
<b>Function descriptions</b>	initialize DCI registers
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dci_struct</b>	address of DCI parameter initialization struct
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* initialize DCI registers */

dci_parameter_struct  dci_struct;

dci_struct.capture_mode = DCI_CAPTURE_MODE_CONTINUOUS;

dci_struct.clock_polarity = DCI_CK_POLARITY_RISING;

dci_struct.hsync_polarity = DCI_HSYNC_POLARITY_LOW;

```

```
dc_i_struct.vsync_polarity = DCI_VSYNC_POLARITY_LOW;
```

```
dc_i_struct.frame_rate = DCI_FRAME_RATE_ALL;
```

```
dc_i_struct.interface_format = DCI_INTERFACE_FORMAT_8BITS;
```

```
dc_i_init(&dc_i_struct);
```

## dc\_i\_enable

The description of dc\_i\_enable is shown as below:

**Table 3-132. Function dc\_i\_enable**

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

Example:

```
/* enable DCI function */
```

```
dc_i_enable();
```

## dc\_i\_disable

The description of dc\_i\_disable is shown as below:

**Table 3-133. Function dc\_i\_disable**

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

Example:

```
/* disable DCI function */

dci_disable();
```

## dci\_capture\_enable

The description of dci\_capture\_enable is shown as below:

**Table 3-134. Function dci\_capture\_enable**

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

Example:

```
/* enable DCI capture */

dci_capture_enable();
```

## dci\_capture\_disable

The description of dci\_capture\_disable is shown as below:

**Table 3-135. Function dci\_capture\_disable**

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

Example:

```
/* disable DCI capture */
```

```
dc_i_capture_disable();
```

### dc\_i\_peg\_enable

The description of dc\_i\_peg\_enable is shown as below:

**Table 3-136. Function dc\_i\_peg\_enable**

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

Example:

```
/* enable DCI jpeg mode */
```

```
dc_i_peg_enable();
```

### dc\_i\_peg\_disable

The description of dc\_i\_peg\_disable is shown as below:

**Table 3-137. Function dc\_i\_peg\_disable**

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

Example:

```
/* disable DCI jpeg mode */
```

dcj\_jpeg\_disable();

## dcj\_crop\_window\_enable

The description of dcj\_crop\_window\_enable is shown as below:

**Table 3-138. Function dcj\_crop\_window\_enable**

<b>Function name</b>	dcj_crop_window_enable
<b>Function prototype</b>	void dcj_crop_window_enable(void);
<b>Function descriptions</b>	enable cropping window function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable cropping window function */
```

```
dcj_crop_window_enable();
```

## dcj\_crop\_window\_disable

The description of dcj\_crop\_window\_disable is shown as below:

**Table 3-139. Function dcj\_crop\_window\_disable**

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

Example:

```
/* disable cropping window function */
```

```
dcj_crop_window_disable();
```



## dc\_i\_crop\_window\_config

The description of dc\_i\_crop\_window\_config is shown as below:

**Table 3-140. Function dc\_i\_crop\_window\_config**

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

Example:

```
/* config DCI cropping window */
```

```
dc_i_crop_window_config (0x800, 0x600, 0x100, 0x100);
```

## dc\_i\_embedded\_sync\_enable

The description of dc\_i\_embedded\_sync\_enable is shown as below:

**Table 3-141. Function dc\_i\_embedded\_sync\_enable**

<b>Function name</b>	dc_i_embedded_sync_enable
<b>Function prototype</b>	void dc_i_embedded_sync_enable(void);
<b>Function descriptions</b>	enable embedded synchronous mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable embedded synchronous mode */
```

```
dc_i_embedded_sync_enable();
```

### dc\_i\_embedded\_sync\_disable

The description of dc\_i\_embedded\_sync\_disable is shown as below:

**Table 3-142. Function dc\_i\_embedded\_sync\_disable**

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

Example:

```
/* disable embedded synchronous mode */
```

```
dc_i_embedded_sync_disable();
```

### dc\_i\_sync\_codes\_config

The description of dc\_i\_sync\_codes\_config is shown as below:

**Table 3-143. Function dc\_i\_sync\_codes\_config**

<b>Function name</b>	dc_i_sync_codes_config
<b>Function prototype</b>	void dc_i_sync_codes_config(uint8_t frame_start, uint8_t line_start, uint8_t line_end, uint8_t frame_end);
<b>Function descriptions</b>	config synchronous codes in embedded synchronous mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>frame_start</b>	frame start code in embedded synchronous mode
<b>Input parameter{in}</b>	
<b>line_start</b>	line start code in embedded synchronous mode
<b>Input parameter{in}</b>	
<b>line_end</b>	line end code in embedded synchronous mode
<b>Input parameter{in}</b>	

<b>frame_end</b>	frame end code in embedded synchronous mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config synchronous codes in embedded synchronous mode */
```

```
dc_i_sync_codes_config(0x10, 0x10, 0x20, 0x20);
```

## dc\_i\_sync\_codes\_unmask\_config

The description of dc\_i\_sync\_codes\_unmask\_config is shown as below:

**Table 3-144. Function dc\_i\_sync\_codes\_unmask\_config**

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

Example:

```
/* config synchronous codes unmask in embedded synchronous mode */
```

```
dc_i_sync_codes_unmask_config(0x10, 0x10, 0x20, 0x20);
```

## dc\_i\_data\_read

The description of dc\_i\_data\_read is shown as below:

**Table 3-145. Function dc\_i\_data\_read**

<b>Function name</b>	dc_i_data_read
----------------------	----------------

<b>Function prototype</b>	uint32_t dci_data_read(void);
<b>Function descriptions</b>	read DCI data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	0x00 – 0xFFFFFFFF

Example:

```
/* read DCI data register */
uint32_t data = dci_data_read();
```

### dc\_i\_flag\_get

The description of dc\_i\_flag\_get is shown as below:

**Table 3-146. Function dc\_i\_flag\_get**

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

Example:

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

## dc\_i\_interrupt\_enable

The description of dc\_i\_interrupt\_enable is shown as below:

**Table 3-147. Function dc\_i\_interrupt\_enable**

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

Example:

```
/* enable specified DCI interrupt */
dc_i_interrupt_enable(DCI_INT_EF);
```

## dc\_i\_interrupt\_disable

The description of dc\_i\_interrupt\_disable is shown as below:

**Table 3-148. Function dc\_i\_interrupt\_disable**

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

-	-
Return value	
-	-

Example:

```
/* disable specified DCI interrupt */
```

```
dc_i_interrupt_disable(DCI_INT_EF);
```

### dc\_i\_interrupt\_flag\_get

The description of dc\_i\_interrupt\_flag\_get is shown as below:

**Table 3-149. Function dc\_i\_interrupt\_flag\_get**

<b>Function name</b>	dc_i_interrupt_flag_get
<b>Function prototype</b>	FlagStatus dc_i_interrupt_flag_get(uint32_t interrupt);
<b>Function descriptions</b>	get specified interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	DCI interrupt
DCI_INT_FLAG_EF	end of frame interrupt flag
DCI_INT_FLAG_OVR	FIFO overrun interrupt flag
DCI_INT_FLAG_ESE	embedded synchronous error interrupt flag
DCI_INT_FLAG_VSYN C	vsync interrupt flag
DCI_INT_FLAG_EL	end of line interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
FlagStatus	FlagStatus: SET or RESET

Example:

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

```
FlagStatus status = dc_i_interrupt_flag_get(DCI_INT_FLAG_EF);
```

### dc\_i\_interrupt\_flag\_clear

The description of dc\_i\_interrupt\_flag\_clear is shown as below:

**Table 3-150. Function dc\_i\_interrupt\_flag\_clear**

<b>Function name</b>	dc_i_interrupt_flag_clear
<b>Function prototype</b>	void dc_i_interrupt_flag_clear(uint32_t interrupt);
<b>Function descriptions</b>	clear specified interrupt flag
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	DCI interrupt
<i>DCI_INT_FLAG_EF</i>	end of frame interrupt flag
<i>DCI_INT_FLAG_OVR</i>	FIFO overrun interrupt flag
<i>DCI_INT_FLAG_ESE</i>	embedded synchronous error interrupt flag
<i>DCI_INT_FLAG_VSYN</i> <i>C</i>	vsync interrupt flag
<i>DCI_INT_FLAG_EL</i>	end of line interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### 3.8.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

**Table 3-151. DMA Registers**

Registers	Descriptions
DMA_INTF0	Interrupt flag register0
DMA_INTF1	Interrupt flag register1
DMA_INTC0	Interrupt flag clear register0
DMA_INTC1	Interrupt flag clear register1
DMA_CHxCTL (x=0..7)	Channel x control register
DMA_CHxCNT (x=0..7)	Channel x counter register
DMA_CHxPADDR (x=0..7)	Channel x peripheral base address register
DMA_CHxM0ADDR	Channel x memory 0 base address register

Registers	Descriptions
(x=0..7)	
DMA_CHxM1ADDR (x=0..7)	Channel x memory 1 base address register
DMA_CHxFCTL	Channel x FIFO control register
DMA_SSTAT	Security status register
DMA_SSC	Security status clear register
DMA_CHxSCTL	Channel x security control register

### 3.8.2. Descriptions of Peripheral functions

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

**Table 3-152. DMA firmware function**

Function name	Function description
dma_deinit	deinitialize DMA a channel registers
dma_single_struct_para_struct_init	initialize the DMA single data mode parameters struct with the default values
dma_multi_struct_para_struct_init	initialize the DMA multi data mode parameters struct with the default values
dma_single_data_mode_init	DMA single data mode initialize
dma_multi_data_mode_init	DMA multi data mode initialize
dma_periph_address_config	set DMA peripheral base address
dma_memory_address_config	set DMA memory base address
dma_transfer_number_config	set the number of remaining data to be transferred by the DMA
dma_transfer_number_get	get the number of remaining data to be transferred by the DMA
dma_priority_config	configure priority level of DMA channel
dma_memory_burst_beats_config	configure transfer burst beats of memory
dma_periph_burst_beats_config	configure transfer burst beats of peripheral
dma_memory_width_config	configure transfer data size of memory
dma_periph_width_config	configure transfer data size of peripheral
dma_memory_address_generation_config	configure next address increasement algorithm of memory
dma_peripheral_address_generation_config	configure next address increasement algorithm of peripheral
dma_circulation_enable	enable DMA circulation mode
dma_circulation_disable	disable DMA circulation mode
dma_channel_enable	enable DMA channel
dma_channel_disable	disable DMA channel
dma_transfer_direction_config	configure the direction of data transfer on the channel
dma_switch_buffer_mode_config	configure DMA switch buffer mode



Function name	Function description
<code>dma_using_memory_get</code>	get DMA using memory
<code>dma_channel_subperipheral_select</code>	select DMA channel peripheral
<code>dma_flow_controller_config</code>	configure DMA flow controller
<code>dma_switch_buffer_mode_enable</code>	enable DMA switch buffer mode
<code>dma_switch_buffer_mode_disable</code>	disable DMA switch buffer mode
<code>dma_round_robin_group_arbitration_enable</code>	enable ultra-high priority channel polling arbitration
<code>dma_round_robin_group_arbitration_disable</code>	disable ultra-high priority channel polling arbitration
<code>dma_fifo_status_get</code>	get DMA FIFO status
<code>dma_flag_get</code>	check the designated flag of a DMA channel is set or not
<code>dma_flag_clear</code>	clear the designated flag of a DMA channel
<code>dma_interrupt_flag_get</code>	check the designated interrupt flag of a DMA channel is set or not
<code>dma_interrupt_flag_clear</code>	clear the designated interrupt flag of a DMA channel
<code>dma_interrupt_enable</code>	enable DMA interrupt
<code>dma_interrupt_disable</code>	disable DMA interrupt
<code>dma_security_config</code>	configure secure attribution of DMA channel
<code>dma_priv_config</code>	configure privileged mode of DMA channel
<code>dma_illegal_access_flag_get</code>	check DMA channel illegal flag is set or not
<code>dma_illegal_access_flag_clear</code>	clear DMA channel illegal flag

## Enum `dma_channel_enum`

**Table 3-153. Enum `dma_channel_enum`**

Member name	Function description
<code>DMA_CH0</code>	DMA Channel0
<code>DMA_CH1</code>	DMA Channel1
<code>DMA_CH2</code>	DMA Channel2
<code>DMA_CH3</code>	DMA Channel3
<code>DMA_CH4</code>	DMA Channel4
<code>DMA_CH5</code>	DMA Channel5
<code>DMA_CH6</code>	DMA Channel6
<code>DMA_CH7</code>	DMA Channel7

## Enum `dma_subperipheral_enum`

**Table 3-154. Enum `dma_subperipheral_enum`**

Member name	Function description
<code>DMA_SUBPERI0</code>	DMA Peripheral 0
<code>DMA_SUBPERI1</code>	DMA Peripheral 1
<code>DMA_SUBPERI2</code>	DMA Peripheral 2

DMA_SUBPERI3	DMA Peripheral 3
DMA_SUBPERI4	DMA Peripheral 4
DMA_SUBPERI5	DMA Peripheral 5
DMA_SUBPERI6	DMA Peripheral 6
DMA_SUBPERI7	DMA Peripheral 7

## Structure dma\_multi\_data\_parameter\_struct

**Table 3-155. Structure dma\_multi\_data\_parameter\_struct**

Member name	Function description
periph_addr	peripheral base address
periph_width	transfer data size of peripheral
periph_inc	peripheral increasing mode
memory0_addr	memory 0 base address
memory_width	transfer data size of memory
memory_inc	memory increasing mode
memory_burst_width	multi data mode enable
periph_burst_width	multi data mode enable
critical_value	FIFO critical
circular_mode	DMA circular mode
direction	channel data transfer direction
number	channel transfer number
priority	channel priority level

## Structure dma\_single\_data\_parameter\_struct

**Table 3-156. Structure dma\_single\_data\_parameter\_struct**

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

## dma\_deinit

The description of dma\_deinit is shown as below:

**Table 3-157. Function dma\_deinit**

<b>Function name</b>	dma_deinit
<b>Function prototype</b>	void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	deinitialize DMA a channel registers
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel is deinitialized, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## **dma\_single\_data\_para\_struct\_init**

The description of dma\_single\_data\_para\_struct\_init is shown as below:

**Table 3-158. Function dma\_single\_data\_para\_struct\_init**

<b>Function name</b>	dma_single_data_para_struct_init
<b>Function prototype</b>	void dma_single_data_para_struct_init(dma_single_data_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize the DMA single data mode parameters struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>init_struct</b>	the address of structure for initialization, the structure members can refer to <a href="#">Table 3-156. Structure dma_single_data_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the parameters struct of DMA with single data mode*/
```

```
dma_single_data_parameter_struct dma_init_struct;
dma_single_data_para_struct_init(&dma_init_struct);
```

### dma\_multi\_data\_para\_struct\_init

The description of dma\_multi\_data\_para\_struct\_init is shown as below:

**Table 3-159. Function dma\_multi\_data\_para\_struct\_init**

<b>Function name</b>	dma_multi_data_para_struct_init
<b>Function prototype</b>	void dma_multi_data_para_struct_init(dma_multi_data_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize the DMA multi data mode parameters struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>init_struct</b>	the address of structure for initialization, the structure members can refer to <a href="#">Table 3-155. Structure dma_multi_data_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the parameters struct of DMA with multi data mode*/
dma_multi_data_parameter_struct dma_init_struct;
dma_multi_data_para_struct_init (&dma_init_struct);
```

### dma\_single\_data\_mode\_init

The description of dma\_single\_data\_mode\_init is shown as below:

**Table 3-160. Function dma\_single\_data\_mode\_init**

<b>Function name</b>	dma_single_data_mode_init
<b>Function prototype</b>	void dma_single_data_mode_init(uint32_t dma_periph, dma_channel_enum channelx, dma_single_data_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize DMA single data mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	specify which DMA channel is deinitialized, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<i>DMA_CHx(x=0..7)</i>	DMA channel selection

Input parameter{in}	
<b>init_struct</b>	the address of structure for initialization, the structure members can refer to <a href="#">Table 3-156. Structure dma_single_data_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* DMA1 channel0 single data mode initialize */
dma_single_data_parameter_struct dma_init_struct;
dma_deinit(DMA1, DMA_CH0);
dma_single_data_para_struct_init(&dma_init_struct);
dma_init_struct.direction = DMA_MEMORY_TO_MEMORY;
dma_init_struct.memory0_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.periph_memory_width = DMA_MEMORY_WIDTH_32BIT;
dma_init_struct.number = TRANSFER_NUM;
dma_init_struct.circular_mode = DMA_CIRCULAR_MODE_DISABLE;
dma_init_struct.periph_addr = (uint32_t)FLASH_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_single_data_mode_init(DMA1, DMA_CH0, &dma_init_struct);

```

### **dma\_multi\_data\_mode\_init**

The description of dma\_multi\_data\_mode\_init is shown as below:

**Table 3-161. Function dma\_multi\_data\_mode\_init**

Function name	dma_multi_data_mode_init
Function prototype	void dma_multi_data_mode_init(uint32_t dma_periph, dma_channel_enum channelx, dma_multi_data_parameter_struct* init_struct);
Function descriptions	initialize DMA multi data mode
Precondition	-
The called functions	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	specify which DMA channel is deinitialized, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
<b>init_struct</b>	the address of structure for initialization, the structure members can refer to

	<a href="#">Table 3-155. Structure dma_multi_data_parameter_struct</a>
	Output parameter{out}
-	-
	Return value
-	-

Example:

```

/* DMA0 channel0 multi data mode initialize */
dma_multi_data_parameter_struct dma_init_parameter;
dma_multi_data_para_struct_init(&dma_init_parameter);
dma_deinit(DMA1, DMA_CH0);
dma_init_parameter.periph_addr = (uint32_t)source;
dma_init_parameter.periph_width = DMA_PERIPH_WIDTH_16BIT;
dma_init_parameter.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_parameter.memory0_addr = (uint32_t)destination;
dma_init_parameter.memory_width = DMA_MEMORY_WIDTH_16BIT;
dma_init_parameter.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_parameter.memory_burst_width = DMA_MEMORY_BURST_4_BEAT;
dma_init_parameter.periph_burst_width = DMA_PERIPH_BURST_4_BEAT;
dma_init_parameter.critical_value = DMA_FIFO_2_WORD;
dma_init_parameter.circular_mode = DMA_CIRCULAR_MODE_DISABLE;
dma_init_parameter.direction = DMA_MEMORY_TO_MEMORY;
dma_init_parameter.number = BUFFER_SIZE;
dma_init_parameter.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_multi_data_mode_init(DMA1,DMA_CH0, &dma_init_parameter);

```

## dma\_periph\_address\_config

The description of dma\_periph\_address\_config is shown as below:

**Table 3-162. Function dma\_periph\_address\_config**

<b>Function name</b>	dma_periph_address_config
<b>Function prototype</b>	void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
<b>Function descriptions</b>	set DMA peripheral base address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	

<b>address</b>	peripheral base address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA0 channel0 peripheral address */

#define USART_DATA_ADDR (USART0 + 0x00000024U)
dma_periph_address_config (DMA0, DMA_CH0, USART_DATA_ADDR);
```

### **dma\_memory\_address\_config**

The description of dma\_memory\_address\_config is shown as below:

**Table 3-163. Function dma\_memory\_address\_config**

<b>Function name</b>	dma_memory_address_config
<b>Function prototype</b>	void dma_memory_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint8_t memory_flag, uint32_t address);
<b>Function descriptions</b>	set DMA memory0 base address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
DMAx(x=0,1)	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>address</b>	memory0 base address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA0 channel0 memory0 address */

uint32_t array[10] = {0};
dma_memory_address_config (DMA0, DMA_CH0, array);
```

### **dma\_transfer\_number\_config**

The description of dma\_transfer\_number\_config is shown as below:

**Table 3-164. Function dma\_transfer\_number\_config**

<b>Function name</b>	dma_transfer_number_config
<b>Function prototype</b>	void dma_transfer_number_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t number);
<b>Function descriptions</b>	set the number of remaining data to be transferred by the DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>number</b>	data transfer number (0x00000000 – 0x0000FFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA0 channel0 transfer number */
#define TRANSFER_NUM                0x400

dma_transfer_number_config(DMA0, DMA_CH0, TRANSFER_NUM);
```

## **dma\_transfer\_number\_get**

The description of dma\_transfer\_number\_get is shown as below:

**Table 3-165. Function dma\_transfer\_number\_get**

<b>Function name</b>	dma_transfer_number_get
<b>Function prototype</b>	uint32_t dma_transfer_number_get(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	get the number of remaining data to be transferred by the DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	



-	-
<b>Return value</b>	
uint32_t	0x00000000 – 0x0000FFFF

Example:

```
/* get channel0 memory0 address */
```

```
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 3-166. Function dma\_priority\_config**

<b>Function name</b>	dma_priority_config
<b>Function prototype</b>	void dma_priority_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priority);
<b>Function descriptions</b>	configure priority level of DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
DMAx(x=0, 1)	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>priority</b>	priority Level of this channel
DMA_PRIORITY_LOW	low priority
DMA_PRIORITY_MEDIUM	medium priority
DMA_PRIORITY_HIGH	high priority
DMA_PRIORITY_ULTRA_HIGH	ultra high priority
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## dma\_memory\_burst\_beats\_config

The description of dma\_memory\_burst\_beats\_config is shown as below:

**Table 3-167. Function dma\_memory\_burst\_beats\_config**

<b>Function name</b>	dma_memory_burst_beats_config
<b>Function prototype</b>	void dma_memory_burst_beats_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t mbeat);
<b>Function descriptions</b>	configure transfer burst beats of memory
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>mbeat</b>	transfer burst beats
<i>DMA_MEMORY_BURST_SINGLE</i>	memory transfer single burst
<i>DMA_MEMORY_BURST_4_BEAT</i>	memory transfer 4-beat burst
<i>DMA_MEMORY_BURST_8_BEAT</i>	memory transfer 8-beat burst
<i>DMA_MEMORY_BURST_16_BEAT</i>	memory transfer 16-beat burst
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA0 channel0 transfer burst beats of memory */
dma_memory_burst_beats_config(DMA0, DMA_CH0, DMA_MEMORY_BURST_8_BEAT);
```

## dma\_periph\_burst\_beats\_config

The description of dma\_periph\_burst\_beats\_config is shown as below:

**Table 3-168. Function dma\_periph\_burst\_beats\_config**

<b>Function name</b>	dma_periph_burst_beats_config
<b>Function prototype</b>	void dma_periph_burst_beats_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t pbeat);

<b>Function descriptions</b>	configure transfer burst beats of peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>pbeat</b>	transfer burst beats
<i>DMA_PERIPH_BURST_SINGLE</i>	peripheral transfer single burst
<i>DMA_PERIPH_BURST_4_BEAT</i>	peripheral transfer 4-beat burst
<i>DMA_PERIPH_BURST_8_BEAT</i>	peripheral transfer 8-beat burst
<i>DMA_PERIPH_BURST_16_BEAT</i>	peripheral transfer 16-beat burst
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA0 channel0 transfer burst beats of peripheral */
```

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

## dma\_memory\_width\_config

The description of dma\_memory\_width\_config is shown as below:

**Table 3-169. Function dma\_memory\_width\_config**

<b>Function name</b>	dma_memory_width_config
<b>Function prototype</b>	void dma_memory_width_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t msize);
<b>Function descriptions</b>	configure transfer data size of memory
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	

<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>msize</b>	transfer data width of memory
<i>DMA_MEMORY_WIDT H_8BIT</i>	transfer data width of memory is 8-bit
<i>DMA_MEMORY_WIDT H_16BIT</i>	transfer data width of memory is 16-bit
<i>DMA_MEMORY_WIDT H_32BIT</i>	transfer data width of memory is 32-bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## **dma\_periph\_width\_config**

The description of dma\_periph\_width\_config is shown as below:

**Table 3-170. Function dma\_periph\_width\_config**

<b>Function name</b>	dma_periph_width_config
<b>Function prototype</b>	void dma_periph_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t psize);
<b>Function descriptions</b>	configure transfer data size of peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>psize</b>	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</i>	transfer data width of peripheral is 32-bit

<i>IDTH_32BIT</i>	
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## **dma\_memory\_address\_generation\_config**

The description of dma\_memory\_address\_generation\_config is shown as below:

**Table 3-171. Function dma\_memory\_address\_generation\_config**

<b>Function name</b>	dma_memory_address_generation_config
<b>Function prototype</b>	void dma_memory_address_generation_config(uint32_t dma_periph, dma_channel_enum channelx, uint8_t generation_algorithm);
<b>Function descriptions</b>	configure memory address generation algorithm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>generation_algorithm</b>	memory address generation algorithm
<i>DMA_MEMORY_INCR EASE_ENABLE</i>	next address of memory is increasing address mode
<i>DMA_MEMORY_INCR EASE_DISABLE</i>	next address of memory is fixed address mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA0 channel0 memory address generation algorithm as memory increase*/
```

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

## dma\_peripheral\_address\_generation\_config

The description of dma\_peripheral\_address\_generation\_config is shown as below:

**Table 3-172. Function dma\_peripheral\_address\_generation\_config**

<b>Function name</b>	dma_peripheral_address_generation_config
<b>Function prototype</b>	void dma_peripheral_address_generation_config(uint32_t dma_periph, dma_channel_enum channelx, uint8_t generation_algorithm);
<b>Function descriptions</b>	configure peripheral address generation algorithm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>generation_algorithm</b>	transfer data width of peripheral
<i>DMA_PERIPH_INCREASE_ENABLE</i>	next address of peripheral is increasing address mode
<i>DMA_PERIPH_INCREASE_DISABLE</i>	next address of peripheral is fixed address mode
<i>DMA_PERIPH_INCREASE_FIX</i>	increasing steps of peripheral address is fixed
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA0 channel0 peripheral address generation algorithm as peripheral increase*/
dma_peripheral_address_generation_config(DMA0, DMA_CH0, DMA_PERIPH_INCREASE_ENABLE);
```

## dma\_circulation\_enable

The description of dma\_circulation\_enable is shown as below:

**Table 3-173. Function dma\_circulation\_enable**

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

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-174. Function dma\_circulation\_disable**

<b>Function name</b>	dma_circulation_disable
<b>Function prototype</b>	void dma_circulation_disable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	disable DMA circulation mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## dma\_channel\_enable

The description of dma\_channel\_enable is shown as below:

**Table 3-175. Function dma\_channel\_enable**

<b>Function name</b>	dma_channel_enable
<b>Function prototype</b>	void dma_channel_enable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	enable DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## dma\_channel\_disable

The description of dma\_channel\_disable is shown as below:

**Table 3-176. Function dma\_channel\_disable**

<b>Function name</b>	dma_channel_disable
<b>Function prototype</b>	void dma_channel_disable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	disable DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	



-	-
Return value	
-	-

Example:

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

## **dma\_transfer\_direction\_config**

The description of dma\_transfer\_direction\_config is shown as below:

**Table 3-177. Function dma\_transfer\_direction\_config**

<b>Function name</b>	dma_transfer_direction_config
<b>Function prototype</b>	void dma_transfer_direction_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t direction);
<b>Function descriptions</b>	configure the direction of data transfer on the channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>direction</b>	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
<i>DMA_MEMORY_TO_MEMORY</i>	read from memory and write to memory
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA0 channel0 transfer direction as peripheral to memory mode*/
dma_transfer_direction_config(DMA0, DMA_CH0, DMA_PERIPHERAL_TO_MEMORY);
```

## dma\_switch\_buffer\_mode\_config

The description of dma\_switch\_buffer\_mode\_config is shown as below:

**Table 3-178. Function dma\_switch\_buffer\_mode\_config**

<b>Function name</b>	dma_switch_buffer_mode_config
<b>Function prototype</b>	void dma_switch_buffer_mode_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t memory1_addr, uint32_t memory_select);
<b>Function descriptions</b>	configure DMA switch buffer mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>memory1_addr</b>	memory1 base address
<b>Input parameter{in}</b>	
<b>memory_select</b>	select memory transfer area
<i>DMA_MEMORY_0</i>	select memory0 as memory transfer area
<i>DMA_MEMORY_1</i>	select memory1 as memory transfer area
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA0 channel0 switch buffer mode*/
```

```
uint32_t mem[32] = {0};
```

```
dma_switch_buffer_mode_config(DMA0, DMA_CH0, mem, DMA_MEMORY_0);
```

## dma\_using\_memory\_get

The description of dma\_using\_memory\_get is shown as below:

**Table 3-179. Function dma\_using\_memory\_get**

<b>Function name</b>	dma_using_memory_get
<b>Function prototype</b>	uint32_t dma_using_memory_get(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	configure DMA switch buffer mode

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>memory1_addr</b>	memory1 base address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	the memory buffer area currently used
<i>DMA_MEMORY_0</i>	memory0 is selected as memory transfer area
<i>DMA_MEMORY_1</i>	memory1 is selected as memory transfer area

Example:

```
/* get DMA0 channel0 transfer buffer currently used */
uint32_t buf_addr = DMA_MEMORY_0;
buf_addr = dma_using_memory_get(DMA0, DMA_CH0);
```

## **dma\_channel\_subperipheral\_select**

The description of dma\_channel\_subperipheral\_select is shown as below:

**Table 3-180. Function dma\_channel\_subperipheral\_select**

<b>Function name</b>	dma_channel_subperipheral_select
<b>Function prototype</b>	void dma_channel_subperipheral_select(uint32_t dma_periph, dma_channel_enum channelx, dma_subperipheral_enum sub_periph);
<b>Function descriptions</b>	DMA channel peripheral select
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>sub_periph</b>	specify DMA channel peripheral
<i>DMA_SUBPERIx(x=0..7)</i>	DMA Peripheral x(x = 0...7)

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
dma_channel_subperipheral_select(DMA0, DMA_CH0, DMA_SUBPERI1);
```

## dma\_flow\_controller\_config

The description of dma\_flow\_controller\_config is shown as below:

**Table 3-181. Function dma\_flow\_controller\_config**

Function name	dma_flow_controller_config
Function prototype	void dma_flow_controller_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t controller);
Function descriptions	configure DMA flow controller
Precondition	-
The called functions	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
Input parameter{in}	
<b>controller</b>	specify DMA flow controller
<i>DMA_FLOW_CONTROLLER_DMA</i>	DMA is the flow controller
<i>DMA_FLOW_CONTROLLER_PERI</i>	peripheral is the flow controller
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA0 channel0 flow controller */
```

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

## dma\_switch\_buffer\_mode\_enable

The description of dma\_switch\_buffer\_mode\_enable is shown as below:

**Table 3-182. Function dma\_switch\_buffer\_mode\_enable**

<b>Function name</b>	dma_switch_buffer_mode_enable
<b>Function prototype</b>	void dma_switch_buffer_mode_enable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	enable DMA switch buffer mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA0 channel0 switch buffer mode */
```

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

## dma\_switch\_buffer\_mode\_disable

The description of dma\_switch\_buffer\_mode\_disable is shown as below:

**Table 3-183. Function dma\_switch\_buffer\_mode\_disable**

<b>Function name</b>	dma_switch_buffer_mode_disable
<b>Function prototype</b>	void dma_switch_buffer_mode_disable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	enable DMA switch buffer mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* disable DMA0 channel0 switch buffer mode */
```

```
dma_switch_buffer_mode_disable (DMA0, DMA_CH0);
```

### **dma\_switch\_buffer\_mode\_disable**

The description of dma\_switch\_buffer\_mode\_disable is shown as below:

**Table 3-184. Function dma\_switch\_buffer\_mode\_disable**

<b>Function name</b>	dma_switch_buffer_mode_disable
<b>Function prototype</b>	void dma_switch_buffer_mode_disable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	disable DMA switch buffer mode
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA switch buffer mode */
```

```
dma_switch_buffer_mode_disable();
```

### **dma\_round\_robin\_group\_arbitration\_enable**

The description of dma\_round\_robin\_group\_arbitration\_enable is shown as below:

**Table 3-185. Function dma\_switch\_buffer\_mode\_disable**

<b>Function name</b>	dma_round_robin_group_arbitration_enable
<b>Function prototype</b>	void dma_round_robin_group_arbitration_enable(uint32_t dma_periph);
<b>Function descriptions</b>	enable ultra-high priority channel polling arbitration
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>dma_periph</b>	DMA peripheral
<i>DMAx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable round-robin group arbitration for "ultra high" priority level channels */
```

```
dma_round_robin_group_arbitration_enable(DMA0);
```

### **dma\_round\_robin\_group\_arbitration\_disable**

The description of dma\_round\_robin\_group\_arbitration\_disable is shown as below:

**Table 3-186. Function dma\_switch\_buffer\_mode\_disable**

<b>Function name</b>	dma_round_robin_group_arbitration_disable
<b>Function prototype</b>	void dma_round_robin_group_arbitration_disable(uint32_t dma_periph);
<b>Function descriptions</b>	disable ultra-high priority channel polling arbitration
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable round-robin group arbitration for "ultra high" priority level channels */
```

```
dma_round_robin_group_arbitration_disable(DMA0);
```

### **dma\_fifo\_status\_get**

The description of dma\_fifo\_status\_get is shown as below:

**Table 3-187. Function dma\_fifo\_status\_get**

<b>Function name</b>	dma_fifo_status_get
<b>Function prototype</b>	uint32_t dma_fifo_status_get(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	get DMA FIFO status
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
Output parameter{out}	
-	-
Return value	
<b>uint32_t</b>	the using memory in FIFO

Example:

```
/* get DMA0 channel0 FIFO status */
```

```
uint32_t fifo_state = 0;
```

```
fifo_state = dma_fifo_status_get(DMA0, DMA_CH0);
```

## dma\_flag\_get

The description of dma\_flag\_get is shown as below:

**Table 3-188. Function dma\_flag\_get**

<b>Function name</b>	dma_flag_get
<b>Function prototype</b>	FlagStatus dma_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	check the designated flag of a DMA channel is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
Input parameter{in}	
<b>flag</b>	specify get which flag
<i>DMA_FLAG_FEE</i>	FIFO error and exception flag
<i>DMA_FLAG_SDE</i>	single data mode exception flag
<i>DMA_FLAG_TAE</i>	transfer access error flag
<i>DMA_FLAG_HTF</i>	half transfer finish flag
<i>DMA_FLAG_FTF</i>	full transfer finish flag
Output parameter{out}	
-	-
Return value	



<b>FlagStatus</b>	SET or RESET
-------------------	--------------

Example:

```
/* check DMA0 channel0 full transfer finish flag set or not */
```

```
FlagStatus state = RESET;
```

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

## **dma\_flag\_clear**

The description of dma\_flag\_clear is shown as below:

**Table 3-189. Function dma\_flag\_clear**

<b>Function name</b>	dma_flag_clear
<b>Function prototype</b>	void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	clear the designated flag of a DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_FLAG_FEE</i>	FIFO error and exception flag
<i>DMA_FLAG_SDE</i>	single data mode exception flag
<i>DMA_FLAG_TAE</i>	transfer access error flag
<i>DMA_FLAG_HTF</i>	half transfer finish flag
<i>DMA_FLAG_FTF</i>	full transfer finish flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear DMA0 channel0 full transfer finish flag */
```

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

## **dma\_interrupt\_flag\_get**

The description of dma\_interrupt\_flag\_get is shown as below:

Table 3-190. 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 int_flag);
Function descriptions	check the designated interrupt flag of a DMA channel is set or not
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0, 1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
Input parameter{in}	
int_flag	specify get which interrupt flag
DMA_INT_FLAG_FEE	FIFO error and exception interrupt flag
DMA_INT_FLAG_SDE	single data mode exception interrupt flag
DMA_INT_FLAG_TAE	transfer access error interrupt flag
DMA_INT_FLAG_HTF	half transfer finish interrupt flag
DMA_INT_FLAG_FTF	full transfer finish interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check DMA0 channel0 full transfer finish interrupt flag set or not */
```

```
FlagStatus state = RESET;
```

```
state = dma_interrupt_flag_get (DMA0, DMA_CH0, DMA_INT_FLAG_FTF);
```

### dma\_interrupt\_flag\_clear

The description of dma\_interrupt\_flag\_clear is shown as below:

Table 3-191. 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 int_flag);
Function descriptions	clear the designated interrupt flag of a DMA channel
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral

<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>int_flag</b>	specify get which interrupt flag
<i>DMA_INT_FLAG_FEE</i>	FIFO error and exception interrupt flag
<i>DMA_INT_FLAG_SDE</i>	single data mode exception interrupt flag
<i>DMA_INT_FLAG_TAE</i>	transfer access error interrupt flag
<i>DMA_INT_FLAG_HTF</i>	half transfer finish interrupt flag
<i>DMA_INT_FLAG_FTF</i>	full transfer finish interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear DMA0 channel0 full transfer finish interrupt flag */
dma_interrupt_flag_clear(DMA0, DMA_CH0, DMA_INT_FLAG_FTF);
```

## **dma\_interrupt\_enable**

The description of dma\_interrupt\_enable is shown as below:

**Table 3-192. Function dma\_interrupt\_enable**

<b>Function name</b>	dma_interrupt_enable
<b>Function prototype</b>	void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
<b>Function descriptions</b>	enable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>source</b>	DMA interrupt source
<i>DMA_INT_FEE</i>	FIFO exception interrupt enable
<i>DMA_INT_SDE</i>	single data mode exception interrupt enable
<i>DMA_INT_TAE</i>	transfer access error interrupt enable
<i>DMA_INT_HTF</i>	half transfer finish interrupt enable

<i>DMA_INT_FTF</i>	full transfer finish interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA0 channel0 full transfer finish interrupt */
```

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

## **dma\_interrupt\_disable**

The description of dma\_interrupt\_disable is shown as below:

**Table 3-193. Function dma\_interrupt\_disable**

<b>Function name</b>	dma_interrupt_disable
<b>Function prototype</b>	void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
<b>Function descriptions</b>	disable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>source</b>	DMA interrupt source
<i>DMA_INT_SDE</i>	single data mode exception interrupt enable
<i>DMA_INT_TAE</i>	transfer access error interrupt enable
<i>DMA_INT_HTF</i>	half transfer finish interrupt enable
<i>DMA_INT_FTF</i>	full transfer finish interrupt enable
<i>DMA_INT_FEE</i>	FIFO exception interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA0 channel0 full transfer finish interrupt */
```

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

## **dma\_security\_config**

The description of dma\_security\_config is shown as below:

**Table 3-194. Function dma\_security\_config**

<b>Function name</b>	dma_security_config
<b>Function prototype</b>	void dma_security_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t security);
<b>Function descriptions</b>	configure security attribution of DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>security</b>	security attribution of this channel
<i>DMA_CHANNEL_SECURE</i>	secure mode
<i>DMA_CHANNEL_NON_SECURE</i>	non-secure mode
<i>DMA_CHANNEL_SOURCE_SECURE</i>	the secure DMA transfer from the source
<i>DMA_CHANNEL_SOURCE_NONSECURE</i>	the non-secure DMA transfer from the source
<i>DMA_CHANNEL_DESTINATION_SECURE</i>	the secure DMA transfer to the destination
<i>DMA_CHANNEL_DESTINATION_NONSECURE</i>	the non-secure DMA transfer to the destination
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA0 channel0 security attribution */
dma_security_config (DMA0, DMA_CH0, DMA_CHANNEL_SECURE);
```

## **dma\_priv\_config**

The description of dma\_priv\_config is shown as below:

**Table 3-195. Function dma\_priv\_config**

<b>Function name</b>	dma_priv_config
<b>Function prototype</b>	void dma_priv_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priv);
<b>Function descriptions</b>	configure privileged mode of DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>priv</b>	privileged mode of this channel
<i>DMA_CHANNEL_PRIV</i>	privileged mode
<i>DMA_CHANNEL_NON PRIV</i>	non-privileged mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA0 channel0 privileged mode */
dma_priv_config(DMA0, DMA_CH0, DMA_CHANNEL_PRIV);
```

## **dma\_illegal\_access\_flag\_get**

The description of dma\_illegal\_access\_flag\_get is shown as below:

**Table 3-196. Function dma\_illegal\_access\_flag\_get**

<b>Function name</b>	dma_illegal_access_flag_get
<b>Function prototype</b>	FlagStatus dma_illegal_access_flag_get(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	check DMA channel illegal flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>

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

Example:

```
/* check DMA0 channel0 illegal access flag is set or not */
```

```
FlagStatus state = RESET;
```

```
state = dma_illegal_access_flag_get (DMA0, DMA_CH0);
```

### **dma\_illegal\_access\_flag\_clear**

The description of dma\_illegal\_access\_flag\_clear is shown as below:

**Table 3-197. Function dma\_illegal\_access\_flag\_clear**

Function name	dma_illegal_access_flag_clear
Function prototype	void dma_illegal_access_flag_clear(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	clear DMA channel illegal flag
Precondition	-
The called functions	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-153. Enum dma_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear DMA0 channel0 illegal access flag */
```

```
dma_illegal_access_flag_clear (DMA0, DMA_CH0);
```

## **3.9. EFUSE**

The Efuse controller has efuse macro that store system paramters. As a non-volatile unit of storage, the bit of efuse macro cannot be restored to 0 once it is programmed to 1. The EFUSE registers are listed in chapter [3.9.1](#), the EFUSE firmware functions are introduced in chapter [3.9.2](#).

### 3.9.1. Descriptions of Peripheral registers

EFUSE registers are listed in the table shown as below:

**Table 3-198. EFUSE Registers**

Registers	Descriptions
EFUSE_CS	control and status register
EFUSE_ADDR	address register
EFUSE_CTL	control register
EFUSE_TZCTL	trustzone control register
EFUSE_FP_CTL	flash protection control register
EFUSE_USER_CTL	user control register
EFUSE_MCU_INIT_DATAx(x = 0...2)	mcu initialization data register x(x = 0...2)
EFUSE_AES_KEYx(x = 0...3)	firmware AES key register x(x = 0...3)
EFUSE_ROTTPK_KEYx(x = 0...7)	RoTPK key register x(x = 0...7)
EFUSE_DPx(x = 0,1)	debug password register x(x = 0,1)
EFUSE_IAK_GSSAx(x = 0...15)	IAK key or GSSA register x(x = 0...15)
EFUSE_PUIDx(x = 0...3)	product UID register x(x = 0...3)
EFUSE_HUK_KEYx(x = 0...3)	HUK key register x(x = 0...3)
EFUSE_RF_DATAx(x = 0...11)	RF data register x(x = 0...11)
EFUSE_USER_DATAx(x = 0...7)	user data register x(x = 0...7)
EFUSE_PRE_TZEN	EFUSE Pre-TrustZone enable register
EFUSE_TZ_BOOT_ADDR	TrustZone boot address register
EFUSE_NTZ_BOOT_ADDR	No-TrustZone boot address register

### 3.9.2. Descriptions of Peripheral functions

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

**Table 3-199. EFUSE firmware function**

Function name	Function description
efuse_read	read EFUSE value
efuse_write	write EFUSE



Function name	Function description
efuse_boot_config	configure boot field
efuse_tz_control_config	configure trustzone control field
efuse_fp_config	configure flash protection field
efuse_mcu_init_data_write	write mcu initialization parameters
efuse_aes_key_write	write AES key
efuse_rotpk_key_write	write ROTPK key
efuse_dp_write	write debug password
efuse_iak_write	write IAK key
efuse_user_data_write	write user data
efuse_software_trustzone_enable	enable trustzone by software
efuse_software_trustzone_disable	disable trustzone by software
efuse_boot_address_get	get boot address information
efuse_flag_get	check EFUSE flag is set or not
efuse_flag_clear	clear EFUSE pending flag
efuse_interrupt_enable	enable EFUSE interrupt
efuse_interrupt_disable	disable EFUSE interrupt
efuse_interrupt_flag_get	check EFUSE interrupt flag is set or not
efuse_interrupt_flag_clear	clear EFUSE pending interrupt flag

## Enum efuse\_flag\_enum

Table 3-200. Enum efuse\_flag\_enum

Member name	Function description
EFUSE_PGIF	programming operation completion flag
EFUSE_RDIF	read operation completion flag
EFUSE_OBERIF	overstep boundary error flag

## Enum efuse\_clear\_flag\_enum

Table 3-201. Enum efuse\_clear\_flag\_enum

Member name	Function description
EFUSE_PGIC	clear programming operation completion flag
EFUSE_RDIC	clear read operation completion flag
EFUSE_OBERIC	clear overstep boundary error flag

## Enum efuse\_int\_enum

Table 3-202. Enum efuse\_int\_enum

Member name	Function description
EFUSE_INTEN_PG	programming operation completion interrupt
EFUSE_INTEN_RD	read operation completion interrupt
EFUSE_INTEN_OBER	overstep boundary error interrupt

## Enum efuse\_int\_flag\_enum

**Table 3-203. Enum efuse\_int\_flag\_enum**

Member name	Function description
EFUSE_INT_PGIF	programming operation completion interrupt flag
EFUSE_INT_RDIF	read operation completion interrupt flag
EFUSE_INT_OBERIF	overstep boundary error interrupt flag

## Enum efuse\_clear\_int\_flag\_enum

**Table 3-204. efuse\_clear\_int\_flag\_enum**

Member name	Function description
EFUSE_INT_PGIC	clear programming operation completion interrupt flag
EFUSE_INT_RDIC	clear read operation completion interrupt flag
EFUSE_INT_OBERIC	clear overstep boundary error interrupt flag

## Enum efuse\_tz\_enum

**Table 3-205. Enum efuse\_tz\_enum**

Member name	Function description
EFUSE_TZ	trustzone enable
EFUSE_N_TZ	trustzone disable

## efuse\_read

The description of efuse\_read is shown as below:

**Table 3-206. Function efuse\_read**

Function name	efuse_read
Function prototype	ErrStatus efuse_read(uint32_t ef_addr, uint32_t size, uint32_t buf[]);
Function descriptions	read EFUSE value
Precondition	-
The called functions	-
Input parameter{in}	
ef_addr	EFUSE address (0x00 – 0xf8)
Input parameter{in}	
size	size of EFUSE (0x01 – 0x40 bytes)
Output parameter{out}	
buf	the buffer for storing read-out EFUSE register value
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* read EFUSE USER DATA value */
```

```
uint32_t buffer[8] = {0};
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_read(0xd8, 32, buffer);
```

### efuse\_write

The description of efuse\_write is shown as below:

**Table 3-207. Function efuse\_write**

<b>Function name</b>	efuse_write
<b>Function prototype</b>	ErrStatus efuse_write(uint32_t ef_addr, uint32_t size, uint32_t buf[]);
<b>Function descriptions</b>	write EFUSE
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ef_addr</b>	EFUSE address (0x00 – 0xf8), when ef_addr is greater than 4, the ef_addr must be an integral multiple of 4
<b>Input parameter{in}</b>	
<b>size</b>	size of EFUSE (0x01 – 0x40 bytes)
<b>Output parameter{out}</b>	
<b>buf</b>	the buffer of data written to EFUSE
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* write EFUSE USER DATA*/
```

```
uint32_t buffer[2] = {0x11223344, 0x55667788};
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_write(0xd8, 8, buffer);
```

### efuse\_boot\_config

The description of efuse\_boot\_config is shown as below:

**Table 3-208. Function efuse\_boot\_config**

<b>Function name</b>	efuse_boot_config
<b>Function prototype</b>	ErrStatus efuse_boot_config(uint32_t size, uint8_t bt_value[]);
<b>Function descriptions</b>	boot configuration
<b>Precondition</b>	-
<b>The called functions</b>	efuse_write
<b>Input parameter{in}</b>	
<b>size</b>	size of data (byte), the size must be 1

Input parameter{in}	
<b>bt_value</b>	the value of boot configuration
Output parameter{out}	
-	-
Return value	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* write Efuse control value */
```

```
uint8_t bt_value[1] = 0x32;
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_boot_config(1, bt_value);
```

## efuse\_tz\_control\_config

The description of efuse\_tz\_control\_config is shown as below:

**Table 3-209. Function efuse\_tz\_control\_config**

<b>Function name</b>	efuse_tz_control_config
<b>Function prototype</b>	ErrStatus efuse_tz_control_config(uint32_t size, uint8_t tz_ctl[]);
<b>Function descriptions</b>	trustzone control configuration
<b>Precondition</b>	-
<b>The called functions</b>	efuse_write
Input parameter{in}	
<b>size</b>	size of data (byte), the size must be 1
Input parameter{in}	
<b>tz_ctl</b>	trustzone control value
Output parameter{out}	
-	-
Return value	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* write Trustzone control value */
```

```
uint8_t tz_ctl[1] = 0x32;
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_tz_control_config(1, tz_ctl);
```

## efuse\_fp\_config

The description of efuse\_fp\_config is shown as below:

Table 3-210. Function efuse\_fp\_config

Function name	efuse_fp_config
Function prototype	ErrStatus efuse_fp_config(uint32_t size, uint8_t fp_value[]);
Function descriptions	flash protection configuration
Precondition	-
The called functions	efuse_write
Input parameter{in}	
size	size of data (byte), the size must be 1
Input parameter{in}	
fp_value	flash protection value
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* write Flash protection value */
uint8_t fp_value[1] = 0x02;
ErrStatus flag = ERROR;
flag = efuse_fp_config(1, fp_value);
```

### efuse\_mcu\_init\_data\_write

The description of efuse\_mcu\_init\_data\_write is shown as below:

Table 3-211. Function efuse\_mcu\_init\_data\_write

Function name	efuse_mcu_init_data_write
Function prototype	ErrStatus efuse_mcu_init_data_write(uint32_t size, uint8_t buf[]);
Function descriptions	write mcu initialization parameters
Precondition	-
The called functions	efuse_write
Input parameter{in}	
size	size of data (byte), the size must be 12
Input parameter{in}	
buf	the buffer of data written to efuse
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* write MCU initialization parameters value */
```

```
uint32_t buffer[3] = {0x11223344, 0x55667788, 0x9900aabb};
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_mcu_inti_data_write(12, (uint8_t *)buffer);
```

### efuse\_aes\_key\_write

The description of efuse\_aes\_key\_write is shown as below:

**Table 3-212. Function efuse\_aes\_key\_write**

<b>Function name</b>	efuse_aes_key_write
<b>Function prototype</b>	ErrStatus efuse_aes_key_write(uint32_t size, uint8_t buf[]);
<b>Function descriptions</b>	write AES key
<b>Precondition</b>	-
<b>The called functions</b>	efuse_write
<b>Input parameter{in}</b>	
<b>size</b>	size of data (byte), the size must be 16
<b>Input parameter{in}</b>	
<b>buf</b>	the buffer of data written to efuse
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* write AES key value */
```

```
uint32_t buffer[4] = {0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff};
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_aes_key_write(4, (uint8_t *)buffer);
```

### efuse\_rotpk\_key\_write

The description of efuse\_rotpk\_key\_write is shown as below:

**Table 3-213. Function efuse\_rotpk\_key\_write**

<b>Function name</b>	efuse_rotpk_key_write
<b>Function prototype</b>	ErrStatus efuse_rotpk_key_write(uint32_t size, uint8_t buf[]);
<b>Function descriptions</b>	write ROTPK key
<b>Precondition</b>	-
<b>The called functions</b>	efuse_write
<b>Input parameter{in}</b>	
<b>size</b>	size of data (byte), the size must be 32
<b>Input parameter{in}</b>	

<b>buf</b>	the buffer of data written to efuse
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* write AES key value */
```

```
uint32_t buffer[8] = {0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff,
                      0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff};
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_rotpk_key_write(32, (uint8_t *)buffer);
```

### efuse\_dp\_write

The description of efuse\_dp\_write is shown as below:

**Table 3-214. Function efuse\_dp\_write**

<b>Function name</b>	efuse_dp_write
<b>Function prototype</b>	ErrStatus efuse_dp_write(uint32_t size, uint8_t buf[]);
<b>Function descriptions</b>	write debug password
<b>Precondition</b>	-
<b>The called functions</b>	efuse_write
<b>Input parameter{in}</b>	
<b>size</b>	size of data (byte), the size must be 8
<b>Input parameter{in}</b>	
<b>buf</b>	the buffer of data written to efuse
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* write Debug password value */
```

```
uint32_t buffer[2] = {0x11223344, 0x55667788};
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_dp_write(8, (uint8_t *)buffer);
```

### efuse\_iak\_write

The description of efuse\_iak\_write is shown as below:

Table 3-215. Function efuse\_iak\_write

Function name	efuse_iak_write
Function prototype	ErrStatus efuse_iak_write(uint32_t size, uint8_t buf[]);
Function descriptions	write IAK key
Precondition	-
The called functions	efuse_write
Input parameter{in}	
size	size of data (byte), the size must be 64
Input parameter{in}	
buf	the buffer of data written to efuse
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* write IAK value */
```

```
uint32_t buffer[16] = {0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff,
                        0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff,
                        0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff,
                        0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff };
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_iak_write(64, (uint8_t *)buffer);
```

### efuse\_user\_data\_write

The description of efuse\_user\_data\_write is shown as below:

Table 3-216. Function efuse\_user\_data\_write

Function name	efuse_user_data_write
Function prototype	ErrStatus efuse_user_data_write(uint32_t size, uint8_t buf[]);
Function descriptions	write user data
Precondition	-
The called functions	efuse_write
Input parameter{in}	
size	size of data (byte), the size must be 32
Input parameter{in}	
buf	the buffer of data written to efuse
Output parameter{out}	
-	-
Return value	



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

Example:

```
/* write User data value */

uint32_t buffer[8] = {0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff
                      0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff};

ErrStatus flag = ERROR;

flag = efuse_user_data_write(32, (uint8_t *)buffer);
```

### efuse\_software\_trustzone\_enable

The description of efuse\_software\_trustzone\_enable is shown as below:

**Table 3-217. Function efuse\_software\_trustzone\_enable**

Function name	efuse_software_trustzone_enable
Function prototype	void efuse_software_trustzone_enable(void);
Function descriptions	enable trustzone by software
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable Trustzone function by software */

efuse_software_trustzone_enable();
```

### efuse\_software\_trustzone\_disable

The description of efuse\_software\_trustzone\_disable is shown as below:

**Table 3-218. Function efuse\_software\_trustzone\_disable**

Function name	efuse_software_trustzone_disable
Function prototype	void efuse_software_trustzone_disable(void);
Function descriptions	disable trustzone by software
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable Trustzone function by software */
```

```
efuse_software_trustzone_disable();
```

## efuse\_boot\_address\_get

The description of efuse\_boot\_address\_get is shown as below:

**Table 3-219. Function efuse\_boot\_address\_get**

Function name	efuse_boot_address_get
Function prototype	uint32_t efuse_boot_address_get(efuse_tz_enum tz);
Function descriptions	get boot address information
Precondition	-
The called functions	-
Input parameter{in}	
tz	specifies the current trustzone state, refer to <a href="#">Table 3-205. Enum efuse_tz_enum</a>
EFUSE_TZ	trustzone enable
EFUSE_N_TZ	trustzone disable
Output parameter{out}	
-	-
Return value	
uint32_t	current boot address (0x0 – 0xFFFFFFFF)

Example:

```
/* get boot address when Trustzone disable */
```

```
uint32_t addr = 0;
```

```
addr = efuse_boot_address_get(EFUSE_N_TZ);
```

## efuse\_flag\_get

The description of efuse\_flag\_get is shown as below:

**Table 3-220. Function efuse\_flag\_get**

Function name	efuse_flag_get
Function prototype	FlagStatus efuse_flag_get(efuse_flag_enum efuse_flag);
Function descriptions	check EFUSE flag is set or not
Precondition	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>efuse_flag</b>	specifies to get a flag, refer to <a href="#">Table 3-200. Enum efuse_flag_enum</a>
<i>EFUSE_PGIF</i>	programming operation completion flag
<i>EFUSE_RDIF</i>	read operation completion flag
<i>EFUSE_OBERIF</i>	overstep boundary error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get EFUSE write operation complete flag status */
```

```
FlagStatus state = efuse_flag_get(EFUSE_PGIF);
```

## efuse\_flag\_clear

The description of efuse\_flag\_clear is shown as below:

**Table 3-221. Function efuse\_flag\_clear**

<b>Function name</b>	efuse_flag_clear
<b>Function prototype</b>	void efuse_flag_clear(efuse_clear_flag_enum efuse_cflag);
<b>Function descriptions</b>	clear EFUSE pending flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>efuse_cflag</b>	specifies to clear a flag, refer to <a href="#">Table 3-201. Enum efuse_clear_flag_enum</a>
<i>EFUSE_PGIC</i>	clear programming operation completion flag
<i>EFUSE_RDIC</i>	clear read operation completion flag
<i>EFUSE_OBERIC</i>	clear overstep boundary error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear EFUSE write operation complete flag status */
```

```
efuse_flag_clear(EFUSE_PGIC);
```

## efuse\_interrupt\_enable

The description of efuse\_interrupt\_enable is shown as below:

Table 3-222. Function efuse\_interrupt\_enable

Function name	efuse_interrupt_enable
Function prototype	void efuse_interrupt_enable(efuse_int_enum source);
Function descriptions	enable EFUSE interrupt
Precondition	-
The called functions	-
Input parameter{in}	
source	specifies an interrupt to enable, refer to <a href="#">Table 3-202. Enum efuse_int_enum</a>
EFUSE_INTEN_PG	programming operation completion interrupt
EFUSE_INTEN_RD	read operation completion interrupt
EFUSE_INTEN_OBER	overstep boundary error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable EFUSE write operation complete interrupt */
```

```
efuse_interrupt_enable(EFUSE_INTEN_PG);
```

### efuse\_interrupt\_disable

The description of efuse\_interrupt\_disable is shown as below:

Table 3-223. Function efuse\_interrupt\_disable

Function name	efuse_interrupt_disable
Function prototype	void efuse_interrupt_disable(efuse_int_enum source);
Function descriptions	disable EFUSE interrupt
Precondition	-
The called functions	-
Input parameter{in}	
source	specifies an interrupt to disable, refer to <a href="#">Table 3-202. Enum efuse_int_enum</a>
EFUSE_INTEN_PG	programming operation completion interrupt
EFUSE_INTEN_RD	read operation completion interrupt
EFUSE_INTEN_OBER	overstep boundary error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable EFUSE write operation complete interrupt */
```

```
efuse_interrupt_disable(EFUSE_INTEN_PG);
```

### efuse\_interrupt\_flag\_get

The description of efuse\_interrupt\_flag\_get is shown as below:

**Table 3-224. Function efuse\_interrupt\_flag\_get**

<b>Function name</b>	efuse_interrupt_flag_get
<b>Function prototype</b>	FlagStatus efuse_interrupt_flag_get(efuse_int_flag_enum int_flag);
<b>Function descriptions</b>	check EFUSE interrupt flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	specifies to get a flag, refer to <a href="#">Table 3-203. Enum efuse_int_flag_enum</a>
<i>EFUSE_INT_PGIF</i>	programming operation completion interrupt flag
<i>EFUSE_INT_RDIF</i>	read operation completion interrupt flag
<i>EFUSE_INT_OBERIF</i>	overstep boundary error interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get EFUSE write operation complete interrupt flag status */
```

```
FlagStatus state = efuse_interrupt_flag_get(EFUSE_INT_PGIF);
```

### efuse\_interrupt\_flag\_clear

The description of efuse\_interrupt\_flag\_clear is shown as below:

**Table 3-225. Function efuse\_interrupt\_flag\_clear**

<b>Function name</b>	efuse_interrupt_flag_clear
<b>Function prototype</b>	void efuse_interrupt_flag_clear(efuse_clear_int_flag_enum int_cflag);
<b>Function descriptions</b>	clear EFUSE pending interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_cflag</b>	specifies to clear a flag, refer to <a href="#">Table 3-203. Enum efuse_int_flag_enum</a>
<i>EFUSE_INT_PGIC</i>	clear programming operation completion interrupt flag
<i>EFUSE_INT_RDIC</i>	clear read operation completion interrupt flag
<i>EFUSE_INT_OBERIC</i>	clear overstep boundary error interrupt flag
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* clear EFUSE write operation complete interrupt flag status */
```

```
efuse_interrupt_flag_clear(EFUSE_INT_PGIC);
```

## 3.10. EXTI

EXTI is the interrupt / event controller in the MCU. It contains up to 29 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.10.1](#), the EXTI firmware functions are introduced in chapter [3.10.2](#).

### 3.10.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

**Table 3-226. EXTI Registers**

Registers	Descriptions
EXTI_INTEN	Interrupt enable register
EXTI_EVEN	Event enable register
EXTI_RTEN	Rising edge trigger enable register
EXTI_FTEN	Falling edge trigger enable register
EXTI_SWIEV	Software interrupt event register
EXTI_PD	Pending register
EXTI_SECCFG	Security configuration register
EXTI_PRIVCFG	Privilege configuration register
EXTI_LOCK	Lock register

### 3.10.2. Descriptions of Peripheral functions

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

**Table 3-227. EXTI firmware function**

Function name	Function description
exti_deinit	deinitialize the EXTI, reset the value of all EXTI registers into initial values
exti_init	initialize the EXTI, enable the configuration of EXTI initialize
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

Function name	Function description
exti_security_enable	enable the security attribution from EXTI line x
exti_security_disable	disable the security attribution from EXTI line x
exti_privilege_enable	enable the privileged access from EXTI line x
exti_privilege_disable	disable the privileged access from EXTI line x
exti_lock_enable	lock EXTI security attribution and privileged access configuration
exti_software_interrupt_enable	enable the EXTI software interrupt event
exti_software_interrupt_disable	disable the EXTI software interrupt event
exti_flag_get	get EXTI line x pending flag
exti_flag_clear	clear EXTI line x pending flag
exti_interrupt_flag_get	get EXTI line x flag when the interrupt flag is set
exti_interrupt_flag_clear	clear EXTI line x pending flag

## Enum exti\_line\_enum

**Table 3-228. 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 (only for GD32W515)
EXTI_21	EXTI line 21
EXTI_22	EXTI line 22
EXTI_23	EXTI line 23
EXTI_24	EXTI line 24
EXTI_25	EXTI line 25

Member name	Function description
EXTI_26	EXTI line 26
EXTI_27	EXTI line 27
EXTI_28	EXTI line 28
EXTI_29	EXTI line 29 (only for GD32F5HC)

## Enum exti\_mode\_enum

**Table 3-229. Enum exti\_mode\_enum**

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

## Enum exti\_trig\_type\_enum

**Table 3-230. 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 3-231. Function exti\_deinit**

Function name	exti_deinit
Function prototype	void exti_deinit(void);
Function descriptions	deinitialize the EXTI, reset the value of all EXTI registers into initial values
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 3-232. Function exti\_init**

<b>Function name</b>	exti_init
<b>Function prototype</b>	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
<b>Function descriptions</b>	initialize the EXTI, enable the configuration of EXTI initialize
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-228. Enum exti_line_enum</a>
<i>EXTI_x</i>	x=0,1,2..29
<b>Input parameter{in}</b>	
<b>mode</b>	EXTI mode, refer to <a href="#">Table 3-229. Enum exti_mode_enum</a>
<i>EXTI_INTERRUPT</i>	interrupt mode
<i>EXTI_EVENT</i>	event mode
<b>Input parameter{in}</b>	
<b>trig_type</b>	trigger type, refer to <a href="#">Table 3-230. Enum exti_trig_type_enum</a>
<i>EXTI_TRIG_RISING</i>	rising edge trigger
<i>EXTI_TRIG_FALLING</i>	falling edge trigger
<i>EXTI_TRIG_BOTH</i>	rising edge and falling edge trigger
<i>EXTI_TRIG_NONE</i>	without rising edge or falling edge trigger
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-233. Function exti\_interrupt\_enable**

<b>Function name</b>	exti_interrupt_enable
<b>Function prototype</b>	void exti_interrupt_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable the interrupts from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-228. Enum exti_line_enum</a>
<i>EXTI_x</i>	x=0,1,2..29
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-234. Function exti\_interrupt\_disable**

<b>Function name</b>	exti_interrupt_disable
<b>Function prototype</b>	void exti_interrupt_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable the interrupts from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-228. Enum exti_line_enum</a>
<i>EXTI_x</i>	x=0,1,2..29
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-235. Function exti\_event\_enable**

<b>Function name</b>	exti_event_enable
<b>Function prototype</b>	void exti_event_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable the events from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-228. Enum exti_line_enum</a>
<i>EXTI_x</i>	x=0,1,2..29
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-236. Function exti\_event\_disable**

<b>Function name</b>	exti_event_disable
<b>Function prototype</b>	void exti_event_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable the events from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-228. Enum exti_line_enum</a>
<i>EXTI_x</i>	x=0,1,2..29
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
exti_event_disable(EXTI_0);
```

### exti\_security\_enable

The description of exti\_security\_enable is shown as below:

**Table 3-237. Function exti\_security\_enable**

<b>Function name</b>	exti_security_enable
<b>Function prototype</b>	void exti_security_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable the security attribution from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-228. Enum exti_line_enum</a>
<i>EXTI_x</i>	x=0,1,2..29
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the security attribution */
```

```
exti_security_enable(EXTI_0);
```

### exti\_security\_disable

The description of exti\_security\_disable is shown as below:

**Table 3-238. Function exti\_security\_disable**

<b>Function name</b>	exti_security_disable
<b>Function prototype</b>	void exti_security_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable the security attribution from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-228. Enum exti_line_enum</a>
<i>EXTI_x</i>	x=0,1,2..29
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the security attribution */
```

```
exti_security_disable(EXTI_0);
```

### exti\_privilege\_enable

The description of exti\_privilege\_enable is shown as below:

**Table 3-239. Function exti\_privilege\_enable**

<b>Function name</b>	exti_privilege_enable
<b>Function prototype</b>	void exti_privilege_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable the privileged access from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-228. Enum exti_line_enum</a>
<i>EXTI_x</i>	x=0,1,2..29
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the privileged access */
```

```
exti_privilege_enable(EXTI_0);
```

### exti\_privilege\_disable

The description of exti\_privilege\_disable is shown as below:

**Table 3-240. Function exti\_privilege\_disable**

<b>Function name</b>	exti_privilege_disable
<b>Function prototype</b>	void exti_privilege_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable the privileged access from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-228. Enum exti_line_enum</a>
<i>EXTI_x</i>	x=0,1,2..29
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the privileged access */
```

```
exti_privilege_disable(EXTI_0);
```

### exti\_lock\_enable

The description of exti\_lock\_enable is shown as below:

**Table 3-241. Function exti\_lock\_enable**

<b>Function name</b>	exti_lock_enable
<b>Function prototype</b>	void exti_lock_enable(void);
<b>Function descriptions</b>	lock EXTI security attribution and privileged access configuration
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

```
/* enable lock EXTI security attribution and privileged access configuration */
```

```
exti_lock_enable();
```

## exti\_software\_interrupt\_enable

The description of exti\_software\_interrupt\_enable is shown as below:

**Table 3-242. Function exti\_software\_interrupt\_enable**

<b>Function name</b>	exti_software_interrupt_enable
<b>Function prototype</b>	void exti_software_interrupt_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable the EXTI software interrupt event
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-228. Enum exti_line_enum</a>
<i>EXTI_x</i>	x=0,1,2..29
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 3-243. Function exti\_software\_interrupt\_disable**

<b>Function name</b>	exti_software_interrupt_disable
<b>Function prototype</b>	void exti_software_interrupt_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable the EXTI software interrupt event
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-228. Enum exti_line_enum</a>

<i>EXTI_x</i>	x=0,1,2..29
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-244. Function exti\_flag\_get**

<b>Function name</b>	exti_flag_get
<b>Function prototype</b>	FlagStatus exti_flag_get(exti_line_enum linex);
<b>Function descriptions</b>	get EXTI line x pending flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-228. Enum exti_line_enum</a>
<i>EXTI_x</i>	x=0,1,2..29
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-245. Function exti\_flag\_clear**

<b>Function name</b>	exti_flag_clear
<b>Function prototype</b>	void exti_flag_clear(exti_line_enum linex);
<b>Function descriptions</b>	clear EXTI line x pending flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-228. Enum exti_line_enum</a>

<i>EXTI_x</i>	x=0,1,2..29
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 3-246. Function exti\_interrupt\_flag\_get**

<b>Function name</b>	exti_interrupt_flag_get
<b>Function prototype</b>	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
<b>Function descriptions</b>	get EXTI line x flag when the interrupt flag is set
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-228. Enum exti_line_enum</a>
<i>EXTI_x</i>	x=0,1,2..29
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	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 3-247. Function exti\_interrupt\_flag\_clear**

<b>Function name</b>	exti_interrupt_flag_clear
<b>Function prototype</b>	void exti_interrupt_flag_clear(exti_line_enum linex);
<b>Function descriptions</b>	clear EXTI line x pending flag
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-228. Enum exti_line_enum</a>



<i>EXTI_x</i>	x=0,1,2..29
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
exti_interrupt_flag_clear(EXTI_0);
```

## 3.11. FMC

There is flash controller and option byte for GD32W51x series. The FMC registers are listed in chapter [3.11.1](#) the FMC firmware functions are introduced in chapter [3.11.2](#).

### 3.11.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

**Table 3-248. FMC Registers**

Registers	Descriptions
FMC_KEY	unlock key register
FMC_OBKEY	option bytes unlock key register
FMC_STAT	status register
FMC_CTL	control register
FMC_ADDR	address register
FMC_OBSTAT	option byte status register
FMC_SECKEY	secure unlock key register
FMC_SECSTAT	secure status register
FMC_SECCTL	secure control register
FMC_SECADDR	secure address register
FMC_OBR	option byte register
FMC_OBUSER	option byte user register
FMC_SECMCFG0	secure mark configuration register 0
FMC_DMP0	secure dedicated mark protection register 0
FMC_OBWRP0	option byte write protection area register 0
FMC_SECMCFG1	secure mark configuration register 1
FMC_DMP1	secure dedicated mark protection register 1
FMC_OBWRP1	option byte write protection area register 1
FMC_SECMCFG2	secure mark configuration register 2
FMC_SECMCFG3	secure mark configuration register 3
FMC_OBR1	option byte register 1

Registers	Descriptions
FMC_NODEC0	NO-RTDEC region register 0
FMC_NODEC1	NO-RTDEC region register 1
FMC_NODEC2	NO-RTDEC region register 2
FMC_NODEC3	NO-RTDEC region register 3
FMC_OFRG	offset region register
FMC_OFVR	offset value register
FMC_DMPCTL	DMP control register
FMC_PRIVCFG	privilege configuration register
FMC_PID	Product ID register

## 3.11.2. Descriptions of Peripheral functions

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

**Table 3-249. FMC firmware function**

Function name	Function description
fmc_unlock	unlock the main FMC operation
fmc_lock	lock the main FMC operation
fmc_page_erase	FMC erase page
fmc_mass_erase	FMC erase whole chip
fmc_word_program	FMC program a word at the corresponding address
fmc_continuous_program	FMC program continuously at the corresponding address
sram1_reset_enable	enable SRAM1 reset automatically function
sram1_reset_disable	disable SRAM1 reset automatically function
fmc_privilege_enable	enable the privileged access
fmc_privilege_disable	disable the privileged access
ob_unlock	unlock the option byte operation
ob_lock	lock the option byte operation
ob_start	send option bytes modification start command
ob_reload	reload option bytes
ob_security_protection_config	configure the option bytes security protection
ob_wdgt_config	configure the option byte for fwdgt/deepsleep/standby
ob_boot_config	configure the option byte for boot0/boot1
ob_trustzone_enable	enable trustzone
ob_trustzone_disable	disable trustzone
ob_user_write	program option bytes USER
ob_write_protection_config	configure write protection pages
ob_secmark_config	configure secure mark pages
ob_dmp_access_enable	enable DMP region access right
ob_dmp_access_disable	disable DMP region access right
ob_dmp_config	configure DMP secure pages
ob_dmp_enable	enable DMP function

Function name	Function description
ob_dmp_disable	disable DMP function
fmc_no_rtdec_config	configure NO-RTDEC pages
fmc_offset_region_config	configure offset region
fmc_offset_value_config	configure offset value
ob_write_protection_get	get the value of option bytes write protection state, only applies to get the status of write/erase protection setting by Efuse
ob_user_get	get the value of option bytes USER
ob_security_protection_flag_get	get option bytes security protection state
ob_trustzone_state_get	get option bytes trustzone state
ob_exist_state_get	get the state of whether the option byte exist or not
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
fmc_interrupt_flag_clear	clear FMC interrupt flag
fmc_ready_wait	check whether FMC is ready or not

## fmc\_state\_enum

Table 3-250. fmc\_state\_enum

enum name	enum description
FMC_READY	the operation has been completed
FMC_BUSY	the operation is in progress
FMC_WPERR	erase/program protection error
FMC_TOERR	timeout error
FMC_OBERR	option bytes error
FMC_SECERR	secure error, only available in secure mode

## fmc\_unlock

The description of fmc\_unlock is shown as below:

Table 3-251. 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 3-252. Function fmc\_lock**

<b>Function name</b>	fmc_lock
<b>Function prototype</b>	void fmc_lock(void);
<b>Function descriptions</b>	lock the main FMC operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
fmc_lock();
```

### fmc\_page\_erase

The description of fmc\_page\_erase is shown as below:

**Table 3-253. Function fmc\_page\_erase**

<b>Function name</b>	fmc_page_erase
<b>Function prototype</b>	fmc_state_enum fmc_page_erase(uint32_t page_address);
<b>Function descriptions</b>	FMC erase page
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>page_address</b>	target page address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-250. fmc_state_enum</a>
-----------------------	--

Example:

```
/* erase page */

fmc_state_enum state;

fmc_unlock();

state = fmc_page_erase ( 0x08004000);
```

### fmc\_mass\_erase

The description of fmc\_mass\_erase is shown as below:

**Table 3-254. Function fmc\_mass\_erase**

<b>Function name</b>	fmc_mass_erase
<b>Function prototype</b>	fmc_state_enum fmc_mass_erase(void );
<b>Function descriptions</b>	erase whole chip
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-250. fmc_state_enum</a>

Example:

```
/* erase the whole chip */

fmc_unlock();

fmc_state_enum state;

state = fmc_mass_erase();
```

### fmc\_word\_program

The description of fmc\_word\_program is shown as below:

**Table 3-255. Function fmc\_word\_program**

<b>Function name</b>	fmc_word_program
<b>Function prototype</b>	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
<b>Function descriptions</b>	program a word at the corresponding address
<b>Precondition</b>	fmc_unlock, fmc_page_erase/fmc_mass_erase
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>address</b>	the address to program
<b>data</b>	word to program(0x00000000 - 0xFFFFFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-250. fmc_state_enum</a>

Example:

```
/* program a word at the corresponding address */

fmc_state_enum state;

fmc_unlock();

fmc_page_erase(0x08004000);

state = fmc_word_program (0x08004000, 0xaabbccdd);
```

### fmc\_continuous\_program

The description of fmc\_continuous\_program is shown as below:

**Table 3-256. Function fmc\_continuous\_program**

<b>Function name</b>	fmc_continuous_program
<b>Function prototype</b>	fmc_state_enum fmc_continuous_program(uint32_t address, uint32_t data[], uint32_t size);
<b>Function descriptions</b>	FMC program data continuously at the corresponding address
<b>Precondition</b>	fmc_unlock, fmc_page_erase/fmc_mass_erase
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>address</b>	address to program, must be 4-byte aligned
<b>Input parameter{in}</b>	
<b>data[]</b>	data buffer to program
<b>Input parameter{in}</b>	
<b>size</b>	data buffer size in bytes, must be 4-byte aligned
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-250. fmc_state_enum</a>

Example:

```
/* program data continuously at the corresponding address */

uint32_t data[16]= {0x01234567, 0x01234567, 0x01234567, 0x01234567, 0x01234567,
0x01234567, 0x01234567, 0x01234567, 0x01234567, 0x01234567, 0x01234567,
0x01234567, 0x01234567, 0x01234567, 0x01234567, 0x01234567};

fmc_state_enum state;
```

```
fmc_unlock();
```

```
state = fmc_continuous_program(0x08004000, data[], 16);
```

## sram1\_reset\_enable

The description of sram1\_reset\_enable is shown as below:

**Table 3-257. Function sram1\_reset\_enable**

<b>Function name</b>	sram1_reset_enable
<b>Function prototype</b>	void sram1_reset_enable(void);
<b>Function descriptions</b>	enable SRAM1 reset automatically function
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SRAM1 reset automatically function */
ob_unlock();
sram1_reset_enable();
```

## sram1\_reset\_disable

The description of sram1\_reset\_disable is shown as below:

**Table 3-258. Function sram1\_reset\_disable**

<b>Function name</b>	sram1_reset_disable
<b>Function prototype</b>	void sram1_reset_disable(void);
<b>Function descriptions</b>	disable SRAM1 reset automatically function
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SRAM1 reset automatically function */
```

```
ob_unlock();
```

```
sram1_reset_disable();
```

## fmc\_privilege\_enable

The description of fmc\_privilege\_enable is shown as below:

**Table 3-259. Function fmc\_privilege\_enable**

<b>Function name</b>	fmc_privilege_enable
<b>Function prototype</b>	void fmc_privilege_enable(void);
<b>Function descriptions</b>	enable the privileged access
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the privileged access */
```

```
fmc_privilege_enable();
```

## fmc\_privilege\_disable

The description of fmc\_privilege\_disable is shown as below:

**Table 3-260. Function fmc\_privilege\_disable**

<b>Function name</b>	fmc_privilege_disable
<b>Function prototype</b>	void fmc_privilege_disable(void);
<b>Function descriptions</b>	disable the privileged access
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:



```
/* disable the privileged access */
```

```
fmc_privilege_disable();
```

## ob\_unlock

The description of ob\_unlock is shown as below:

**Table 3-261. Function ob\_unlock**

<b>Function name</b>	ob_unlock
<b>Function prototype</b>	void ob_unlock(void);
<b>Function descriptions</b>	unlock the option byte operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* unlock the option byte operation */
```

```
ob_unlock();
```

## ob\_lock

The description of ob\_lock is shown as below:

**Table 3-262. Function ob\_lock**

<b>Function name</b>	ob_lock
<b>Function prototype</b>	void ob_lock(void);
<b>Function descriptions</b>	lock the option byte operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock the option byte operation */
```

ob\_lock();

## ob\_start

The description of ob\_start is shown as below:

**Table 3-263. Function ob\_start**

<b>Function name</b>	ob_start
<b>Function prototype</b>	void ob_start(void);
<b>Function descriptions</b>	send option bytes modification start command
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* program option bytes USER data */
```

```
ob_unlock( );
```

```
ob_user_write(0xFFFF);
```

```
ob_start();
```

## ob\_reload

The description of ob\_reload is shown as below:

**Table 3-264. Function ob\_reload**

<b>Function name</b>	ob_reload
<b>Function prototype</b>	void ob_reload(void);
<b>Function descriptions</b>	reload option bytes
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* program option bytes USER data */
```

```
ob_unlock();
```

```
ob_user_write(0xFFFF);
```

```
ob_start();
```

```
ob_reload();
```

### ob\_security\_protection\_config

The description of ob\_security\_protection\_config is shown as below:

**Table 3-265. Function ob\_security\_protection\_config**

<b>Function name</b>	ob_security_protection_config
<b>Function prototype</b>	fmc_state_enum ob_security_protection_config(uint8_t ob_spc);
<b>Function descriptions</b>	configure security protection
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ob_spc</b>	specify security protection
<i>FMC_NSPC</i>	no security protection
<i>FMC_SPC_P0_5</i>	security protection level 0.5
<i>FMC_SPC_P1</i>	security protection level 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-250. fmc_state_enum</a>

Example:

```
/* configure low security protection */
```

```
ob_unlock();
```

```
ob_security_protection_config (FMC_NSPC);
```

```
ob_start();
```

### ob\_wdgt\_config

The description of ob\_wdgt\_config is shown as below:

**Table 3-266. Function ob\_wdgt\_config**

<b>Function name</b>	ob_wdgt_config
<b>Function prototype</b>	void ob_wdgt_config(uint32_t ob_fwdgt, uint32_t ob_deepsleep, uint32_t ob_stdby)
<b>Function descriptions</b>	configure the option byte for fwdgt/deepsleep/standby (API_ID(0x0010U))

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ob_fwdgt</b>	option byte watchdog value
<i>OB_FWDGT_SW</i>	software free watchdog
<i>OB_FWDGT_HW</i>	hardware free watchdog
<b>Input parameter{in}</b>	
<b>ob_deepsleep</b>	option byte deepsleep reset value
<i>OB_DEEPSLEEP_NRS</i> <i>T</i>	no reset when entering deepsleep mode
<i>OB_DEEPSLEEP_RST</i>	generate a reset instead of entering deepsleep mode
<b>Input parameter{in}</b>	
<b>ob_stdby</b>	option byte standby reset value
<i>OB_STDBY_NRST</i>	no reset when entering standby mode
<i>OB_STDBY_RST</i>	generate a reset instead of entering standby mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure wdgt */
```

```
ob_unlock();
```

```
ob_wdgt_config(OB_FWDGT_SW, OB_DEEPSLEEP_NRST, OB_STDBY_NRST);
```

```
ob_start();
```

### ob\_boot\_config

The description of ob\_boot\_config is shown as below:

**Table 3-267. Function ob\_boot\_config**

<b>Function name</b>	ob_boot_config
<b>Function prototype</b>	void ob_boot_config(uint32_t swboot0, uint32_t boot0, uint32_t swboot1, uint32_t boot1)
<b>Function descriptions</b>	configure the option byte for boot0/boot1 (API_ID(0x0011U))
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>swboot0</b>	SWBOOT0 option bits
<i>OB_BOOT0_NBOOT0</i>	Select nBOOT0 as BOOT0 signal
<i>OB_BOOT0_PC8</i>	Select PC8 pin as BOOT0 signal
<b>Input parameter{in}</b>	

<b>boot0</b>	BOOT0 option bits (only valid when swboot0 is OB_BOOT0_NBOOT0)
<i>OB_BOOT0_1</i>	BOOT0 is 1
<i>OB_BOOT0_0</i>	BOOT0 is 0
<b>Input parameter{in}</b>	
<b>swboot1</b>	SWBOOT1 option bit
<i>OB_BOOT1_NBOOT1</i>	Select nBOOT1 as BOOT1 signal
<i>OB_BOOT1_PB2</i>	Select PB2 pin as BOOT1 signal
<b>Input parameter{in}</b>	
<b>boot1</b>	BOOT1 option bits (only valid when swboot1 is OB_BOOT1_NBOOT1)
<i>OB_BOOT1_1</i>	BOOT1 is 1
<i>OB_BOOT1_0</i>	BOOT1 is 0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure boot */
```

```
ob_unlock();
```

```
ob_boot_config(OB_BOOT0_NBOOT0,      OB_BOOT0_1,      OB_BOOT1_NBOOT1,
OB_BOOT1_1);
```

```
ob_start();
```

## ob\_trustzone\_enable

The description of ob\_trustzone\_enable is shown as below:

**Table 3-268. Function ob\_trustzone\_enable**

<b>Function name</b>	ob_trustzone_enable
<b>Function prototype</b>	fmc_state_enum ob_trustzone_enable(void);
<b>Function descriptions</b>	enable trustzone
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-250. fmc_state_enum</a>

Example:

```
/* enable trustzone */
```

```
ob_unlock();
```

```
ob_trustzone_enable();
```

### ob\_trustzone\_disable

The description of ob\_trustzone\_disable is shown as below:

**Table 3-269. Function ob\_trustzone\_disable**

<b>Function name</b>	ob_trustzone_disable
<b>Function prototype</b>	ErrStatus ob_trustzone_disable(void);
<b>Function descriptions</b>	disable trustzone
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	ob_security_protection_flag_get
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
ErrStatus	ERROR or SUCCESS

Example:

```
/* disable trustzone */
```

```
ErrStatus state;
```

```
ob_unlock();
```

```
state = ob_trustzone_disable();
```

### ob\_user\_write

The description of ob\_user\_write is shown as below:

**Table 3-270. Function ob\_user\_write**

<b>Function name</b>	ob_user_write
<b>Function prototype</b>	fmc_state_enum ob_user_write(uint16_t ob_user);
<b>Function descriptions</b>	program option bytes USER
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
ob_user	option bytes user value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, refer to <a href="#">Table 3-250. fmc_state_enum</a>

Example:

```
/* program option bytes USER data */

ob_unlock();

ob_user_write(0xFFFF);

ob_start();
```

### ob\_write\_protection\_config

The description of ob\_write\_protection\_config is shown as below:

**Table 3-271. Function ob\_write\_protection\_config**

<b>Function name</b>	ob_write_protection_config
<b>Function prototype</b>	fmc_state_enum ob_write_protection_config(uint32_t wrp_spag, uint32_t wrp_epag, uint32_t wrp_register_index);
<b>Function descriptions</b>	configure write protection pages
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wrp_spag</b>	start page of write protection area, 0~512
<b>Input parameter{in}</b>	
<b>wrp_epag</b>	end page of write protection area, 0~512
<b>Input parameter{in}</b>	
<b>wrp_register_index</b>	FMC_OBWRPx register index
<i>OBWRP_INDEX0</i>	option byte write protection area register 0
<i>OBWRP_INDEX1</i>	option byte write protection area register 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-250. fmc_state_enum</a>

Example:

```
/* configure write protection pages */

ob_unlock();

ob_write_protection_config(WRP_REGION_SPAGE, WRP_REGION_EPAGE, OBWRP_INDEX0);

ob_start();
```

### ob\_secmark\_config

The description of ob\_secmark\_config is shown as below:

Table 3-272. Function ob\_secmark\_config

<b>Function name</b>	ob_secmark_config
<b>Function prototype</b>	void ob_secmark_config(uint32_t secm_spage, uint32_t secm_epage, uint32_t secm_register_index);
<b>Function descriptions</b>	configure secure mark pages
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>secm_spage</b>	start page of secure mark area, 0~0x03FF
<b>Input parameter{in}</b>	
<b>secm_epage</b>	end page of secure mark area, 0~0x03FF
<b>Input parameter{in}</b>	
<b>secm_register_index</b>	secure mark register index
<i>SECM_INDEX0</i>	secure mark configuration register 0
<i>SECM_INDEX1</i>	secure mark configuration register 1
<i>SECM_INDEX2</i>	secure mark configuration register 2
<i>SECM_INDEX3</i>	secure mark configuration register 3
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure secure mark pages */
```

```
ob_unlock();
```

```
ob_secmark_config(SECM_REGION_SPAGE,SECM_REGION_EPAGE,SECM_INDEX3);
```

### ob\_dmp\_access\_enable

The description of ob\_dmp\_access\_enable is shown as below:

Table 3-273. Function ob\_dmp\_access\_enable

<b>Function name</b>	ob_dmp_access_enable
<b>Function prototype</b>	void ob_dmp_access_enable(uint32_t DMP_register_index);
<b>Function descriptions</b>	enable DMP region access right
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dmp_register_index</b>	DMP region configuration register index
<i>DMP_INDEX0</i>	secure DMP configuration register 0
<i>DMP_INDEX1</i>	secure DMP configuration register 1
<b>Output parameter{out}</b>	
-	-



Return value	
-	-

Example:

```
/* enable DMP region access right */
```

```
ob_dmp_access_enable(DMP_INDEX0);
```

### ob\_dmp\_access\_disable

The description of ob\_dmp\_access\_disable is shown as below:

**Table 3-274. Function ob\_dmp\_access\_disable**

Function name	ob_dmp_access_disable
Function prototype	void ob_dmp_access_disable(uint32_t DMP_register_index);
Function descriptions	disable DMP region access right
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
dmp_register_index	DMP region configuration register index
DMP_INDEX0	secure DMP configuration register 0
DMP_INDEX1	secure DMP configuration register 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMP region access right */
```

```
ob_unlock();
```

```
ob_dmp_access_disable(DMP_INDEX0);
```

### ob\_dmp\_config

The description of ob\_dmp\_config is shown as below:

**Table 3-275. Function ob\_dmp\_config**

Function name	ob_dmp_config
Function prototype	ErrStatus ob_dmp_config(uint32_t dmp_epage, uint32_t dmp_register_index);
Function descriptions	configure DMP pages
Precondition	ob_unlock
The called functions	-
Input parameter{in}	

<b>dmp_epage</b>	end page of secure DMP mark area 0~0x03FF
<b>Input parameter{in}</b>	
<b>dmp_register_index</b>	DMP region register index
<i>DMP_INDEX0</i>	secure DMP configuration register 0
<i>DMP_INDEX1</i>	secure DMP configuration register 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* configure DMP pages */

ob_unlock();

ob_dmp_config(DMP_REGION_EPAGE, DMP_INDEX0);
```

### ob\_dmp\_enable

The description of ob\_dmp\_enable is shown as below:

**Table 3-276. Function ob\_dmp\_enable**

<b>Function name</b>	ob_dmp_enable
<b>Function prototype</b>	fmc_state_enum ob_dmp_enable(uint32_t dmp_register_index);
<b>Function descriptions</b>	enable DMP function
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dmp_register_index</b>	DMP region register index
<i>DMP_INDEX0</i>	secure DMP configuration register 0
<i>DMP_INDEX1</i>	secure DMP configuration register 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-250. fmc_state_enum</a>

Example:

```
/* enable DMP function */

ob_unlock();

ob_dmp_enable(DMP_INDEX0);
```

### ob\_dmp\_disable

The description of ob\_dmp\_disable is shown as below:

**Table 3-277. Function ob\_dmp\_disable**

<b>Function name</b>	ob_dmp_disable
<b>Function prototype</b>	fmc_state_enum ob_dmp_disable(uint32_t dmp_register_index);
<b>Function descriptions</b>	disable DMP function
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dmp_register_index</b>	DMP region register index
<i>DMP_INDEX0</i>	secure DMP configuration register 0
<i>DMP_INDEX1</i>	secure DMP configuration register 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-250. fmc_state_enum</a>

Example:

```
/* disable DMP function */
ob_unlock();
ob_dmp_disable(DMP_INDEX0);
```

## fmc\_no\_rtdec\_config

The description of fmc\_no\_rtdec\_config is shown as below:

**Table 3-278. Function fmc\_no\_rtdec\_config**

<b>Function name</b>	fmc_no_rtdec_config
<b>Function prototype</b>	void fmc_no_rtdec_config(uint32_t nodec_spage, uint32_t nodec_epage, uint32_t nodec_register_index);
<b>Function descriptions</b>	configure NO-RTDEC pages
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>nodec_spage</b>	start page of NO-RTDEC area, 0~0x03FF
<b>Input parameter{in}</b>	
<b>nodec_epage</b>	end page of NO-RTDEC area, 0~0x03FF
<b>Input parameter{in}</b>	
<b>nodec_register_index</b>	NO-RTDEC region register index
<i>NODEC_INDEX0</i>	NO-RTDEC region register 0
<i>NODEC_INDEX1</i>	NO-RTDEC region register 1
<i>NODEC_INDEX2</i>	NO-RTDEC region register 2
<i>NODEC_INDEX3</i>	NO-RTDEC region register 3
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure NO-RTDEC pages */
```

```
ob_unlock();
```

```
fmc_no_rtdec_config(NODEC_REGION_SPAGE, NODEC_REGION_EPAGE, NODEC_IN
DEX0);
```

### fmc\_offset\_region\_config

The description of fmc\_offset\_region\_config is shown as below:

**Table 3-279. Function fmc\_offset\_region\_config**

<b>Function name</b>	fmc_offset_region_config
<b>Function prototype</b>	void fmc_offset_region_config(uint32_t of_spage, uint32_t of_epage);
<b>Function descriptions</b>	configure offset region
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>of_spage</b>	start page of offset region, 0~0x1FFF
<b>Input parameter{in}</b>	
<b>of_epage</b>	end page of offset region, 0~0x1FFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure offset region */
```

```
ob_unlock();
```

```
fmc_offset_region_config(SOURCE_START_PAGE, SOURCE_END_PAGE);
```

### fmc\_offset\_value\_config

The description of fmc\_offset\_value\_config is shown as below:

**Table 3-280. Function fmc\_offset\_value\_config**

<b>Function name</b>	fmc_offset_value_config
<b>Function prototype</b>	void fmc_offset_value_config(uint32_t of_value);
<b>Function descriptions</b>	configure offset value
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-

Input parameter{in}	
of_value	offset value, 0~0x1FFF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure offset value */
```

```
ob_unlock();
```

```
fmc_offset_value_config(PAGE_OFFSET_VALUE);
```

### ob\_write\_protection\_get

The description of ob\_write\_protection\_get is shown as below:

**Table 3-281. Function ob\_write\_protection\_get**

Function name	ob_write_protection_get
Function prototype	FlagStatus ob_write_protection_get(void);
Function descriptions	get write protection state, only applies to get the status of write/erase protection setting by EFUSE
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the FMC write protection status */
```

```
FlagStatus status;
```

```
status = ob_write_protection_get();
```

### ob\_user\_get

The description of ob\_user\_get is shown as below:

**Table 3-282. Function ob\_user\_get**

Function name	ob_user_get
Function prototype	uint16_t ob_user_get(void);
Function descriptions	get the value of option bytes USER

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	0x0000~0xFFFF

Example:

```
/* get the value of option bytes USER */
```

```
uint16_t user_value;
```

```
user_value = ob_user_get();
```

### ob\_security\_protection\_flag\_get

The description of ob\_security\_protection\_flag\_get is shown as below:

**Table 3-283. Function ob\_security\_protection\_flag\_get**

<b>Function name</b>	ob_security_protection_flag_get
<b>Function prototype</b>	FlagStatus ob_security_protection_flag_get(uint32_t spc_state);
<b>Function descriptions</b>	get the FMC option bytes security protection state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spc_state</b>	security protection level
<i>OB_FLAG_NSPC</i>	option bytes security protection level 0
<i>OB_FLAG_SPC0_5</i>	option bytes security protection level 0.5
<i>OB_FLAG_SPC1</i>	option bytes security protection level 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check FMC option bytes security protection level 1 is set or not */
```

```
FlagStatus status;
```

```
status = ob_security_protection_flag_get(OB_FLAG_SPC1);
```

### ob\_trustzone\_state\_get

The description of ob\_trustzone\_state\_get is shown as below:

Table 3-284. Function ob\_trustzone\_state\_get

Function name	ob_trustzone_state_get
Function prototype	FlagStatus ob_trustzone_state_get(void);
Function descriptions	get trustzone state
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get trustzone state */
```

```
FlagStatus status;
```

```
status = ob_trustzone_state_get();
```

### ob\_exist\_state\_get

The description of ob\_exist\_state\_get is shown as below:

Table 3-285. Function ob\_exist\_state\_get

Function name	ob_exist_state_get
Function prototype	FlagStatus ob_exist_state_get(void);
Function descriptions	get the state of whether the option byte exist or not
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the state of whether the option byte exist or not */
```

```
FlagStatus status;
```

```
status = ob_exist_state_get();
```

## fmc\_flag\_get

The description of fmc\_flag\_get is shown as below:

**Table 3-286. Function fmc\_flag\_get**

<b>Function name</b>	fmc_flag_get
<b>Function prototype</b>	FlagStatus fmc_flag_get(uint32_t flag);
<b>Function descriptions</b>	check flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	FMC flag
<i>FMC_FLAG_BUSY</i>	FMC busy flag bit
<i>FMC_FLAG_OBERR</i>	FMC option bytes error flag bit
<i>FMC_FLAG_WPERR</i>	FMC erase/program protection error flag bit
<i>FMC_FLAG_END</i>	FMC end of operation flag bit
<i>FMC_FLAG_SECERR</i>	FMC secure error flag bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check the FMC_FLAG_END flag set or not*/
```

```
FlagStatus flag;
```

```
flag = fmc_flag_get(FMC_FLAG_END);
```

## fmc\_flag\_clear

The description of fmc\_flag\_clear is shown as below:

**Table 3-287. Function fmc\_flag\_clear**

<b>Function name</b>	fmc_flag_clear
<b>Function prototype</b>	void fmc_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear the FMC flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	FMC flag
<i>FMC_FLAG_OBERR</i>	FMC option bytes error flag bit
<i>FMC_FLAG_WPERR</i>	FMC erase/program protection error flag bit
<i>FMC_FLAG_END</i>	FMC end of operation flag bit
<i>FMC_FLAG_SECERR</i>	FMC secure error flag bit



Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the FMC_FLAG_END flag */
fmc_flag_clear(FMC_FLAG_END);
```

## fmc\_interrupt\_enable

The description of fmc\_interrupt\_enable is shown as below:

**Table 3-288. Function fmc\_interrupt\_enable**

Function name	fmc_interrupt_enable
Function prototype	void fmc_interrupt_enable (uint32_t interrupt);
Function descriptions	enable FMC interrupt
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
interrupt	the FMC interrupt source
<i>FMC_INT_END</i>	enable FMC end of program interrupt
<i>FMC_INT_ERR</i>	enable FMC error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable FMC interrupt */
fmc_unlock();
fmc_interrupt_enable(FMC_INT_END);
```

## fmc\_interrupt\_disable

The description of fmc\_interrupt\_disable is shown as below:

**Table 3-289. Function fmc\_interrupt\_disable**

Function name	fmc_interrupt_disable
Function prototype	void fmc_interrupt_disable(uint32_t interrupt);
Function descriptions	disable interrupt of fmc
Precondition	fmc_unlock
The called functions	-

Input parameter{in}	
<b>interrupt</b>	the FMC interrupt source
<i>FMC_INT_END</i>	disable FMC end of program interrupt
<i>FMC_INT_ERR</i>	disable FMC error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable FMC interrupt */
fmc_unlock();
fmc_interrupt_disable(FMC_INT_END);
```

### fmc\_interrupt\_flag\_get

The description of fmc\_interrupt\_flag\_get is shown as below:

**Table 3-290. Function fmc\_interrupt\_flag\_get**

<b>Function name</b>	fmc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus fmc_interrupt_flag_get(uint32_t flag);
<b>Function descriptions</b>	get FMC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>flag</b>	the FMC interrupt flag
<i>FMC_INT_FLAG_WPE</i> <i>RR</i>	FMC secure/non-secure erase/program protection error interrupt flag
<i>FMC_INT_FLAG_END</i>	FMC secure/non-secure end of operation interrupt flag
<i>FMC_INT_FLAG_SEC</i> <i>ERR</i>	FMC secure error flag
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check operation interrupt flag is set or not */
FlagStatus flag;
flag = fmc_interrupt_flag_get(FMC_INT_FLAG_END);
```

## fmc\_interrupt\_flag\_clear

The description of fmc\_interrupt\_flag\_clear is shown as below:

**Table 3-291. Function fmc\_interrupt\_flag\_clear**

<b>Function name</b>	fmc_interrupt_flag_clear
<b>Function prototype</b>	void fmc_interrupt_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear FMC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	the FMC interrupt flag
<i>FMC_INT_FLAG_WPE</i> <i>RR</i>	FMC secure/non-secure erase/program protection error interrupt flag
<i>FMC_INT_FLAG_END</i>	FMC secure/non-secure end of operation interrupt flag
<i>FMC_INT_FLAG_SEC</i> <i>ERR</i>	FMC secure error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear operation interrupt flag */
fmc_interrupt_flag_clear(FMC_INT_FLAG_END);
```

## fmc\_ready\_wait

The description of fmc\_ready\_wait is shown as below:

**Table 3-292. Function fmc\_ready\_wait**

<b>Function name</b>	fmc_ready_wait
<b>Function prototype</b>	fmc_state_enum fmc_ready_wait(uint32_t timeout)
<b>Function descriptions</b>	check whether FMC is ready or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timeout</b>	timeout count
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* wait for FMC ready */
```

```
fmc_state_enum state = fmc_ready_wait(0x1000);
```

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

### 3.12.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

**Table 3-293. FWDGT Registers**

Registers	Descriptions
FWDGT_CTL	Control register
FWDGT_PSC	Prescaler register
FWDGT_RLD	Reload register
FWDGT_STAT	Status register
FWDGT_WND	window register

### 3.12.2. Descriptions of Peripheral functions

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

**Table 3-294. 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 FWDGT counter
fwdgt_prescaler_value_config	configure the FWDGT counter prescaler value
fwdgt_reload_value_config	configure the FWDGT counter reload value
fwdgt_window_value_config	configure the FWDGT counter window value
fwdgt_counter_reload	reload the counter of FWDGT
fwdgt_config	configure counter reload value, and prescaler divider value
fwdgt_flag_get	get flag state of FWDGT

## fwdgt\_write\_enable

The description of fwdgt\_write\_enable is shown as below:

**Table 3-295. Function fwdgt\_write\_enable**

<b>Function name</b>	fwdgt_write_enable
<b>Function prototype</b>	void fwdgt_write_enable(void);
<b>Function descriptions</b>	enable write access to FWDGT_PSC and FWDGT_RLD
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-296. Function fwdgt\_write\_disable**

<b>Function name</b>	fwdgt_write_disable
<b>Function prototype</b>	void fwdgt_write_disable(void);
<b>Function descriptions</b>	disable write access to FWDGT_PSC and FWDGT_RLD
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-297. Function fwdgt\_enable**

<b>Function name</b>	fwdgt_enable
<b>Function prototype</b>	void fwdgt_enable(void);
<b>Function descriptions</b>	start the FWDGT counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-298. Function fwdgt\_prescaler\_value\_config**

<b>Function name</b>	fwdgt_prescaler_value_config
<b>Function prototype</b>	ErrStatus fwdgt_prescaler_value_config(uint16_t prescaler_value);
<b>Function descriptions</b>	configure the FWDGT counter prescaler value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>prescaler_value</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

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 3-299. Function fwdgt\_reload\_value\_config**

<b>Function name</b>	fwdgt_reload_value_config
<b>Function prototype</b>	ErrStatus fwdgt_reload_value_config(uint16_t reload_value);
<b>Function descriptions</b>	configure the FWDGT counter reload value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>reload_value</b>	specify reload value(0x0000 - 0x0FFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* configure the FWDGT counter reload value */

fwdgt_reload_value_config(625);
```

### fwdgt\_window\_value\_config

The description of fwdgt\_window\_value\_config is shown as below:

**Table 3-300. Function fwdgt\_window\_value\_config**

<b>Function name</b>	fwdgt_window_value_config
<b>Function prototype</b>	ErrStatus fwdgt_window_value_config(uint16_t window_value);
<b>Function descriptions</b>	configure the FWDGT counter window value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>window_value</b>	window_value, specify window value(0x0000 - 0x0FFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR / SUCCESS

Example:

```
/* set FWDGT window value to 0xFFFF */
```

```
ErrStatus flag;
```

```
flag = fwdgt_window_value_config(0xFFFF);
```

## fwdgt\_counter\_reload

The description of fwdgt\_counter\_reload is shown as below:

**Table 3-301. Function fwdgt\_counter\_reload**

<b>Function name</b>	fwdgt_counter_reload
<b>Function prototype</b>	void fwdgt_counter_reload(void);
<b>Function descriptions</b>	reload the counter of FWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reload FWDGT counter */
```

```
fwdgt_counter_reload ( );
```

## fwdgt\_config

The description of fwdgt\_config is shown as below:

**Table 3-302. Function fwdgt\_config**

<b>Function name</b>	fwdgt_config
<b>Function prototype</b>	ErrStatus fwdgt_config(uint16_t reload_value, uint8_t prescaler_div);
<b>Function descriptions</b>	configure counter reload value, and prescaler divider value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>reload_value</b>	specify reload value(0x0000 - 0x0FFF)
<b>Input parameter{in}</b>	
<b>prescaler_div</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* configure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
```

```
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

### fwdgt\_flag\_get

The description of fwdgt\_flag\_get is shown as below:

**Table 3-303. Function fwdgt\_flag\_get**

<b>Function name</b>	fwdgt_flag_get
<b>Function prototype</b>	FlagStatus fwdgt_flag_get(uint16_t flag);
<b>Function descriptions</b>	get flag state of FWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* test if a prescaler value update is on going */
```

```
FlagStatus status;
```

```
status = fwdgt_flag_get (FWDGT_FLAG_PUD);
```

## 3.13. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.13.1](#), the GPIO firmware functions are introduced in chapter [3.13.2](#).

### 3.13.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

**Table 3-304. GPIO Registers**

Registers	Descriptions
GPIOx_CTL	GPIO port control register
GPIOx_OMODE	GPIO port output mode register
GPIOx_OSPD	GPIO port output speed register
GPIOx_PUD	GPIO port pull-up/pull-down register
GPIOx_ISTAT	GPIO port input status register
GPIOx_OCTL	GPIO port output control register
GPIOx_BOP	GPIO port bit operation register
GPIOx_LOCK	GPIO port configuration lock register
GPIOx_AFSEL0	GPIO alternate function selected register 0
GPIOx_AFSEL1	GPIO alternate function selected register 1
GPIOx_BC	GPIO bit clear register
GPIOx_TG	GPIO port bit toggle register
GPIOx_SCFG	GPIO secure configuration register

### 3.13.2. Descriptions of Peripheral functions

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

**Table 3-305. GPIO firmware function**

Function name	Function description
gpio_deinit	reset GPIO port
gpio_mode_set	set GPIO mode
gpio_output_options_set	set GPIO output type and speed
gpio_bit_set	set GPIO pin bit
gpio_bit_reset	reset GPIO pin bit
gpio_bit_write	write data to the specified GPIO pin
gpio_port_write	write data to the specified GPIO port
gpio_input_bit_get	get GPIO pin input status
gpio_input_port_get	get GPIO port input status
gpio_output_bit_get	get GPIO pin output status
gpio_output_port_get	get GPIO port output status
gpio_af_set	set GPIO alternate function
gpio_pin_lock	lock GPIO pin bit
gpio_bit_toggle	toggle GPIO pin status
gpio_port_toggle	toggle GPIO port status
gpio_bit_set_sec_cfg	configure GPIO pin bit secure configuration bit status to set
gpio_bit_reset_sec_cfg	configure GPIO pin bit secure configuration bit status to reset

Function name	Function description
gpio_sec_cfg_bit_get	get GPIO pin secure configuration bit status

## gpio\_deinit

The description of gpio\_deinit is shown as below:

**Table 3-306. 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	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset GPIOA */
gpio_deinit(GPIOA);
```

## gpio\_mode\_set

The description of gpio\_mode\_set is shown as below:

**Table 3-307. Function gpio\_mode\_set**

Function name	gpio_mode_set
Function prototype	void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin);
Function descriptions	set GPIO mode
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
Input parameter{in}	
mode	gpio pin mode
GPIO_MODE_INPUT	input mode

<i>GPIO_MODE_OUTPUT</i> <i>T</i>	output mode
<i>GPIO_MODE_AF</i>	alternate function mode
<i>GPIO_MODE_ANALOG</i> <i>G</i>	analog mode
<b>Input parameter{in}</b>	
<b>pull_up_down</b>	gpio pin with pull-up or pull-down resistor
<i>GPIO_PUPD_NONE</i>	floating mode, no pull-up and pull-down resistors
<i>GPIO_PUPD_PULLUP</i>	with pull-up resistor
<i>GPIO_PUPD_PULLDOWN</i> <i>WN</i>	with pull-down resistor
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure PA0 as input mode with pullup */
```

```
gpio_mode_set(GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_PULLUP, GPIO_PIN_0);
```

### gpio\_output\_options\_set

The description of gpio\_output\_options\_set is shown as below:

**Table 3-308. Function gpio\_output\_options\_set**

<b>Function name</b>	gpio_output_options_set
<b>Function prototype</b>	void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);
<b>Function descriptions</b>	set GPIO output type and speed
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
<b>Input parameter{in}</b>	
<b>otype</b>	gpio pin output mode
<i>GPIO_OTYPE_PP</i>	push pull mode
<i>GPIO_OTYPE_OD</i>	open drain mode

Input parameter{in}	
<b>speed</b>	gpio pin output max speed
<i>GPIO_OSPEED_2MHZ</i>	output max speed 2MHz
<i>GPIO_OSPEED_10MHZ</i>	output max speed 10MHz
<i>GPIO_OSPEED_25MHZ</i>	output max speed 25MHz
<i>GPIO_OSPEED_166MHZ</i>	output max speed 166MHz
Input parameter{in}	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x (x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure PA0 as push pull mode */
```

```
gpio_output_options_set(GPIOA, GPIO_OTYPE_PP, GPIO_OSPEED_2MHZ,
GPIO_PIN_0);
```

## gpio\_bit\_set

The description of gpio\_bit\_set is shown as below:

**Table 3-309. Function gpio\_bit\_set**

<b>Function name</b>	gpio_bit_set
<b>Function prototype</b>	void gpio_bit_set(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	set GPIO pin bit
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
Input parameter{in}	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* set PA0 */
```

```
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

## gpio\_bit\_reset

The description of gpio\_bit\_reset is shown as below:

**Table 3-310. Function gpio\_bit\_reset**

Function name	gpio_bit_reset
Function prototype	void gpio_bit_reset(uint32_t gpio_periph,uint32_t pin);
Function descriptions	reset GPIO pin bit
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
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 3-311. 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}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
Input parameter{in}	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Input parameter{in}	
<b>bit_value</b>	SET or RESET
<i>RESET</i>	clear the port pin
<i>SET</i>	set the port pin
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write 1 to PA0 */
```

```
gpio_bit_write(GPIOA, GPIO_PIN_0, SET);
```

## gpio\_port\_write

The description of gpio\_port\_write is shown as below:

**Table 3-312. Function gpio\_port\_write**

<b>Function name</b>	gpio_port_write
<b>Function prototype</b>	void gpio_port_write(uint32_t gpio_periph,uint16_t data);
<b>Function descriptions</b>	write data to the specified GPIO port
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
Input parameter{in}	
<b>data</b>	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 3-313. Function gpio\_input\_bit\_get**

<b>Function name</b>	gpio_input_bit_get
<b>Function prototype</b>	FlagStatus gpio_input_bit_get(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	get GPIO pin input status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get status of PA0 */
```

```
FlagStatus 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 3-314. Function gpio\_input\_port\_get**

<b>Function name</b>	gpio_input_port_get
<b>Function prototype</b>	uint16_t gpio_input_port_get(uint32_t gpio_periph);
<b>Function descriptions</b>	get GPIO all pins input status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
<b>Output parameter{out}</b>	



-	-
<b>Return value</b>	
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 3-315. Function gpio\_output\_bit\_get**

<b>Function name</b>	gpio_output_bit_get
<b>Function prototype</b>	FlagStatus gpio_output_bit_get(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	get GPIO pin output status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get output status of PA0 */
```

```
FlagStatus bit_state;
```

```
bit_state = gpio_output_bit_get(GPIOA, GPIO_PIN_0);
```

### gpio\_output\_port\_get

The description of gpio\_output\_port\_get is shown as below:

**Table 3-316. Function gpio\_output\_port\_get**

<b>Function name</b>	gpio_output_port_get
<b>Function prototype</b>	uint16_t gpio_output_port_get(uint32_t gpio_periph);

<b>Function descriptions</b>	get GPIO all pins output status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	0x0000-0xFFFF

Example:

```
/* get output value of Port A */
```

```
uint16_t port_state;
```

```
port_state = gpio_output_port_get(GPIOA);
```

## gpio\_af\_set

The description of gpio\_af\_set is shown as below:

**Table 3-317. Function gpio\_af\_set**

<b>Function name</b>	gpio_af_set
<b>Function prototype</b>	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);
<b>Function descriptions</b>	set GPIO alternate function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
<b>Input parameter{in}</b>	
<b>alt_func_num</b>	GPIO pin af function, please refer to specific device datasheet
<i>GPIO_AF_0</i>	alternate function 0
<i>GPIO_AF_1</i>	alternate function 1
<i>GPIO_AF_2</i>	alternate function 2
<i>GPIO_AF_3</i>	alternate function 3
<i>GPIO_AF_4</i>	alternate function 4
<i>GPIO_AF_5</i>	alternate function 5
<i>GPIO_AF_6</i>	alternate function 6
<i>GPIO_AF_7</i>	alternate function 7
<i>GPIO_AF_8</i>	alternate function 8
<i>GPIO_AF_9</i>	alternate function 9

<i>GPIO_AF_10</i>	alternate function 10
<i>GPIO_AF_11</i>	alternate function 11
<i>GPIO_AF_12</i>	alternate function 12
<i>GPIO_AF_13</i>	alternate function 13
<i>GPIO_AF_14</i>	alternate function 14
<i>GPIO_AF_15</i>	alternate function 15
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*set PA0 alternate function 0 */
```

```
gpio_af_set(GPIOA, GPIO_AF_0, GPIO_PIN_0);
```

## gpio\_pin\_lock

The description of gpio\_pin\_lock is shown as below:

**Table 3-318. Function gpio\_pin\_lock**

<b>Function name</b>	gpio_pin_lock
<b>Function prototype</b>	void gpio_pin_lock(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	lock GPIO pin bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock PA0 */
```

```
gpio_pin_lock(GPIOA, GPIO_PIN_0);
```

## gpio\_bit\_toggle

The description of gpio\_bit\_toggle is shown as below:

**Table 3-319. Function gpio\_bit\_toggle**

<b>Function name</b>	gpio_bit_toggle
<b>Function prototype</b>	void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	toggle GPIO pin status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	GPIO_PIN_ALL
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* toggle PA0 */
```

```
gpio_bit_toggle(GPIOA, GPIO_PIN_0);
```

## gpio\_port\_toggle

The description of gpio\_port\_toggle is shown as below:

**Table 3-320. Function gpio\_port\_toggle**

<b>Function name</b>	gpio_port_toggle
<b>Function prototype</b>	void gpio_port_toggle(uint32_t gpio_periph);
<b>Function descriptions</b>	toggle GPIO port status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* toggle GPIOA */
```

```
gpio_port_toggle(GPIOA);
```

## gpio\_bit\_set\_sec\_cfg

The description of gpio\_bit\_set\_sec\_cfg is shown as below:

**Table 3-321. Function gpio\_bit\_set\_sec\_cfg**

Function name	gpio_bit_set_sec_cfg
Function prototype	void gpio_bit_set_sec_cfg(uint32_t gpio_periph, uint32_t pin);
Function descriptions	configure GPIO pin bit secure configuration bit status to set
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	GPIO_PIN_ALL
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure GPIOA pin 0 secure configuration bit status to set */
```

```
gpio_bit_set_sec_cfg(GPIOA, GPIO_PIN_0);
```

## gpio\_bit\_reset\_sec\_cfg

The description of gpio\_bit\_reset\_sec\_cfg is shown as below:

**Table 3-322. Function gpio\_bit\_reset\_sec\_cfg**

Function name	gpio_bit_reset_sec_cfg
Function prototype	void gpio_bit_reset_sec_cfg(uint32_t gpio_periph, uint32_t pin);
Function descriptions	configure GPIO pin bit secure configuration bit status to reset

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	GPIO_PIN_ALL
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure GPIOA pin 0 secure configuration bit status to reset */
```

```
gpio_bit_reset_sec_cfg(GPIOA, GPIO_PIN_0);
```

### gpio\_sec\_cfg\_bit\_get

The description of gpio\_sec\_cfg\_bit\_get is shown as below:

**Table 3-323. Function gpio\_sec\_cfg\_bit\_get**

<b>Function name</b>	gpio_sec_cfg_bit_get
<b>Function prototype</b>	FlagStatus gpio_sec_cfg_bit_get(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	get GPIO pin secure configuration bit status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GD32W515: GPIOx(x = A,B,C); GD32F5HC: GPIOx(x = A,B,C,D)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get GPIOA pin0 secure configuration bit status */
```

```
FlagStatus sec_cfg_bit_state = gpio_sec_cfg_bit_get(GPIOA, GPIO_PIN_0);
```

## 3.14. HAU

The HASH Acceleration Unit supports acceleration of SHA-1, SHA-224, SHA-256, MD5 algorithm and the HMAC (keyed-hash message authentication code) algorithm. The HAU registers are listed in chapter [3.14.1](#). the HAU firmware functions are introduced in chapter [3.14.2](#).

### 3.14.1. Descriptions of Peripheral registers

HAU registers are listed in the table shown as below:

**Table 3-324. HAU Registers**

Registers	Descriptions
HAU_CTL	control register
HAU_DI	data input register
HAU_CFG	configuration register
HAU_DO0	data output register 0
HAU_DO1	data output register 1
HAU_DO2	data output register 2
HAU_DO3	data output register 3
HAU_DO4	data output register 4
HAU_DO5	data output register 5
HAU_DO6	data output register 6
HAU_DO7	data output register 7
HAU_INTEN	interrupt enable register
HAU_STAT	status and interrupt flag register
HAU_CTXSx (x = 0..53)	context switch register

### 3.14.2. Descriptions of Peripheral functions

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

**Table 3-325. HAU firmware function**

Function name	Function description
hau_deinit	reset the HAU peripheral
hau_init	initialize the HAU peripheral parameters
hau_init_struct_para_init	initialize the struct hau_initpara
hau_reset	reset the HAU processor core
hau_last_word_validbits_num_config	configure the number of valid bits in last word of the message
hau_data_write	write data to the IN FIFO

Function name	Function description
hau_inifo_words_num_get	return the number of words already written into the IN FIFO
hau_digest_read	read the message digest result
hau_digest_calculation_enable	enable digest calculation
hau_multiple_single_dma_config	configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer or not
hau_dma_enable	enable the HAU DMA interface
hau_dma_disable	disable the HAU DMA interface
hau_context_struct_para_init	initialize the struct context
hau_context_save	save the HAU peripheral context
hau_context_restore	restore the HAU peripheral context
hau_hash_sha_1	calculate digest using SHA1 in HASH mode
hau_hmac_sha_1	calculate digest using SHA1 in HMAC mode
hau_hash_sha_224	calculate digest using SHA224 in HASH mode
hau_hmac_sha_224	calculate digest using SHA224 in HMAC mode
hau_hash_sha_256	calculate digest using SHA256 in HASH mode
hau_hmac_sha_256	calculate digest using SHA256 in HMAC mode
hau_hash_md5	calculate digest using MD5 in HASH mode
hau_hmac_md5	calculate digest using MD5 in HMAC mode
hau_flag_get	get the HAU flag status
hau_flag_clear	clear the HAU flag status
hau_interrupt_enable	enable the HAU interrupts
hau_interrupt_disable	disable the HAU interrupts
hau_interrupt_flag_get	get the HAU interrupt flag status
hau_interrupt_flag_clear	clear the HAU interrupt flag status

## Structure hau\_init\_parameter\_struct

Table 3-326. Structure hau\_init\_parameter\_struct

Member name	Function description
algo	algorithm selection: HAU_ALGO_SHA1, HAU_ALGO_SHA224, HAU_ALGO_SHA256, HAU_ALGO_MD5
mode	HAU mode selection: HAU_MODE_HASH, HAU_MODE_HMAC
datatype	data type mode: HAU_SWAPPING_32BIT, HAU_SWAPPING_16BIT, HAU_SWAPPING_8BIT, HAU_SWAPPING_1BIT
keytype	key length mode: HAU_KEY_SHORTER_64, HAU_KEY_LONGGER_64

## Structure hau\_digest\_parameter\_struct

Table 3-327. Structure hau\_digest\_parameter\_struct

Member name	Function description
-------------	----------------------



out[8]

message digest result 0-7

## Structure hau\_context\_parameter\_struct

**Table 3-328. Structure hau\_context\_parameter\_struct**

Member name	Function description
hau_ctl_bak	backup of HAU_CTL register
hau_cfg_bak	backup of HAU_CFG register
hau_inten_bak	backup of HAU_INTEN register
hau_ctxs_bak[54]	backup of HAU_CTXSx registers

## hau\_deinit

The description of hau\_deinit is shown as below:

**Table 3-329. Function hau\_deinit**

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

Example:

```
/* reset the HAU peripheral */
```

```
hau_deinit();
```

## hau\_init

The description of hau\_init is shown as below:

**Table 3-330. Function hau\_init**

Function name	hau_init
Function prototype	void hau_init(hau_init_parameter_struct* initpara);
Function descriptions	initialize the HAU peripheral parameters
Precondition	-
The called functions	-
Input parameter{in}	
initpara	the HAU peripheral parameters, refer to structure <a href="#">Table 3-326. Structure hau_init_parameter_struct</a>

Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize the HAU peripheral parameters */

hau_init_parameter_struct hau_initpara;

hau_init_struct_para_init(&hau_initpara);

hau_initpara.algo = algo;

hau_initpara.mode = HAU_MODE_HMAC;

hau_initpara.datatype = HAU_SWAPPING_8BIT;

if(key_len > 64U){
    hau_initpara.keytype = HAU_KEY_LONGGER_64;
}else{
    hau_initpara.keytype = HAU_KEY_SHORTER_64;
}

hau_init(&hau_initpara);

```

### hau\_init\_struct\_para\_init

The description of hau\_init\_struct\_para\_init is shown as below:

**Table 3-331. Function hau\_init\_struct\_para\_init**

Function name	hau_init_struct_para_init
Function prototype	void hau_init_struct_para_init(hau_init_parameter_struct* initpara);
Function descriptions	initialize the sturct hau_initpara
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
initpara	the HAU peripheral parameters, refer to structure <a href="#">Table 3-326. Structure hau_init_parameter_struct</a>
Return value	
-	-

Example:

```

/* initialize the HAU peripheral parameters */

```

```
hau_init_parameter_struct hau_initpara;
```

```
hau_init_struct_para_init(&hau_initpara);
```

## hau\_reset

The description of hau\_reset is shown as below:

**Table 3-332. Function hau\_reset**

<b>Function name</b>	hau_reset
<b>Function prototype</b>	void hau_reset(void);
<b>Function descriptions</b>	reset the HAU processor core
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset the HAU processor core */
hau_reset();
```

## hau\_last\_word\_validbits\_num\_config

The description of hau\_last\_word\_validbits\_num\_config is shown as below:

**Table 3-333. Function hau\_last\_word\_validbits\_num\_config**

<b>Function name</b>	hau_last_word_validbits_num_config
<b>Function prototype</b>	void hau_last_word_validbits_num_config(uint32_t valid_num);
<b>Function descriptions</b>	configure the number of valid bits in last word of the message
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>valid_num</b>	number of valid bits in last word of the message(0x00 – 0x1F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the number of valid bits in last word of the message */
```

```
hau_last_word_validbits_num_config(0x10);
```

## hau\_data\_write

The description of hau\_data\_write is shown as below:

**Table 3-334. Function hau\_data\_write**

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

Example:

```
/* write data to the IN FIFO */
```

```
hau_data_write(0x10);
```

## hau\_infifo\_words\_num\_get

The description of hau\_infifo\_words\_num\_get is shown as below:

**Table 3-335. Function hau\_infifo\_words\_num\_get**

<b>Function name</b>	hau_infifo_words_num_get
<b>Function prototype</b>	uint32_t hau_infifo_words_num_get(void);
<b>Function descriptions</b>	return the number of words already written into the IN FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	0x0 – 0xFFFFFFFF

Example:

```
/* return the number of words already written into the IN FIFO */
```

```
uint32_t num;
```

```
num = hau_infifo_words_num_get();
```

## hau\_digest\_read

The description of hau\_digest\_read is shown as below:

**Table 3-336. Function hau\_digest\_read**

<b>Function name</b>	hau_digest_read
<b>Function prototype</b>	void hau_digest_read(hau_digest_parameter_struct* digestpara);
<b>Function descriptions</b>	read the message digest result
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
digestpara	the message digest result, refer to structure <a href="#">Table 3-327. Structure hau_digest_parameter_struct</a>
<b>Return value</b>	
-	-

Example:

```
/* read the message digest result */
hau_digest_parameter_struct digestpara;
hau_digest_read(&digestpara);
```

## hau\_digest\_calculation\_enable

The description of hau\_digest\_calculation\_enable is shown as below:

**Table 3-337. Function hau\_digest\_calculation\_enable**

<b>Function name</b>	hau_digest_calculation_enable
<b>Function prototype</b>	void hau_digest_calculation_enable(void);
<b>Function descriptions</b>	enable digest calculation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable digest calculation */
```

```
hau_digest_calculation_enable();
```

## hau\_multiple\_single\_dma\_config

The description of hau\_multiple\_single\_dma\_config is shown as below:

**Table 3-338. Function hau\_multiple\_single\_dma\_config**

<b>Function name</b>	hau_multiple_single_dma_config
<b>Function prototype</b>	void hau_multiple_single_dma_config(uint32_t multi_single);
<b>Function descriptions</b>	configure single or multiple DMA is used, and digest calculation at the end of a DMA transfer or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>multi_single</b>	Multiple or single
<i>SINGLE_DMA_AUTO_DIGEST</i>	message padding and message digest calculation at the end of a DMA transfer
<i>MULTIPLE_DMA_NO_DIGEST</i>	multiple DMA transfers needed and CALEN bit is not automatically set at the end of a DMA transfer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer or not */
```

```
hau_multiple_single_dma_config(SINGLE_DMA_AUTO_DIGEST);
```

## hau\_dma\_enable

The description of hau\_dma\_enable is shown as below:

**Table 3-339. Function hau\_dma\_enable**

<b>Function name</b>	hau_dma_enable
<b>Function prototype</b>	void hau_dma_enable(void);
<b>Function descriptions</b>	enable the HAU DMA interface
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* enable the HAU DMA interface */
```

```
hau_dma_enable();
```

## hau\_dma\_disable

The description of hau\_dma\_disable is shown as below:

**Table 3-340. Function hau\_dma\_disable**

<b>Function name</b>	hau_dma_disable
<b>Function prototype</b>	void hau_dma_disable(void);
<b>Function descriptions</b>	disable the HAU DMA interface
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the HAU DMA interface */
```

```
hau_dma_disable();
```

## hau\_context\_struct\_para\_init

The description of hau\_context\_struct\_para\_init is shown as below:

**Table 3-341. Function hau\_context\_struct\_para\_init**

<b>Function name</b>	hau_context_struct_para_init
<b>Function prototype</b>	void hau_context_struct_para_init(hau_context_parameter_struct* context);
<b>Function descriptions</b>	initialize the struct context
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>context</b>	the context, refer to structure <a href="#">Table 3-328. Structure hau_context_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the struct context */

hau_context_parameter_struct context_para;

hau_context_struct_para_init(&context_para);
```

## hau\_context\_save

The description of hau\_context\_save is shown as below:

**Table 3-342. Function hau\_context\_save**

<b>Function name</b>	hau_context_save
<b>Function prototype</b>	void hau_context_save(hau_context_parameter_struct* context_save);
<b>Function descriptions</b>	save the HAU peripheral context
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
context_save	the HAU peripheral context, refer to structure <a href="#">Table 3-328. Structure hau_context_parameter_struct</a>
<b>Return value</b>	
-	-

Example:

```
hau_context_parameter_struct context_para;

hau_context_struct_para_init(&context_para);

/* save HAU context */

hau_context_save(&context_para);
```

## hau\_context\_restore

The description of hau\_context\_restore is shown as below:

**Table 3-343. Function hau\_context\_restore**

<b>Function name</b>	hau_context_restore
<b>Function prototype</b>	void hau_context_restore(hau_context_parameter_struct* context_restore);
<b>Function descriptions</b>	restore the HAU peripheral context
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
context_restore	the HAU peripheral context, refer to structure <a href="#">Table 3-328. Structure hau_context_parameter_struct</a>



Output parameter{out}	
-	-
Return value	
-	-

Example:

```

hau_context_parameter_struct context_para;

hau_context_struct_para_init(&context_para);

hau_context_save(&context_para);

/* restore HAU context */

hau_context_restore(&context_para);

```

## hau\_hash\_sha\_1

The description of hau\_hash\_sha\_1 is shown as below:

**Table 3-344. Function hau\_hash\_sha\_1**

Function name	hau_hash_sha_1
Function prototype	ErrStatus hau_hash_sha_1(uint8_t *input, uint32_t in_length, uint8_t output[20]);
Function descriptions	calculate digest using SHA1 in HASH mode
Precondition	-
The called functions	-
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```

/* calculate digest using SHA1 in HASH mode */

ErrStatus status = hau_hash_sha_1(&input, 0x10, output[0]);

```

## hau\_hmac\_sha\_1

The description of hau\_hmac\_sha\_1 is shown as below:

**Table 3-345. Function hau\_hmac\_sha\_1**

Function name	hau_hmac_sha_1
---------------	----------------

<b>Function prototype</b>	ErrStatus hau_hmac_sha_1(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[20]);
<b>Function descriptions</b>	calculate digest using SHA1 in HMAC mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>key</b>	pointer to the key used for HMAC
<b>Input parameter{in}</b>	
<b>keysize</b>	length of the key used for HMAC
<b>Input parameter{in}</b>	
<b>input</b>	pointer to the input buffer
<b>Input parameter{in}</b>	
<b>in_length</b>	length of the input buffer
<b>Output parameter{out}</b>	
<b>output</b>	the result digest
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA1 in HMAC mode */
```

```
ErrStatus status = hau_hmac_sha_1(&key, 0x10, &input, 0x10, output[0]);
```

## hau\_hash\_sha\_224

The description of hau\_hash\_sha\_224 is shown as below:

**Table 3-346. Function hau\_hash\_sha\_224**

<b>Function name</b>	hau_hash_sha_224
<b>Function prototype</b>	ErrStatus hau_hash_sha_224(uint8_t *input, uint32_t in_length, uint8_t output[28]);
<b>Function descriptions</b>	calculate digest using SHA224 in HASH mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>input</b>	pointer to the input buffer
<b>Input parameter{in}</b>	
<b>in_length</b>	length of the input buffer
<b>Output parameter{out}</b>	
<b>output</b>	the result digest
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA224 in HASH mode */
```

```
ErrStatus status = hau_hash_sha_224 (&input, 0x10, output[0]);
```

## hau\_hmac\_sha\_224

The description of hau\_hmac\_sha\_224 is shown as below:

**Table 3-347. Function hau\_hmac\_sha\_224**

<b>Function name</b>	hau_hmac_sha_224
<b>Function prototype</b>	ErrStatus hau_hmac_sha_224(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[28]);
<b>Function descriptions</b>	calculate digest using SHA224 in HMAC mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>key</b>	pointer to the key used for HMAC
<b>Input parameter{in}</b>	
<b>keysize</b>	length of the key used for HMAC
<b>Input parameter{in}</b>	
<b>input</b>	pointer to the input buffer
<b>Input parameter{in}</b>	
<b>in_length</b>	length of the input buffer
<b>Output parameter{out}</b>	
<b>output</b>	the result digest
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA224 in HMAC mode */
```

```
ErrStatus status = hau_hmac_sha_224 (&key, 0x10, &input, 0x10, output[0]);
```

## hau\_hash\_sha\_256

The description of hau\_hash\_sha\_256 is shown as below:

**Table 3-348. Function hau\_hash\_sha\_256**

<b>Function name</b>	hau_hash_sha_256
<b>Function prototype</b>	ErrStatus hau_hash_sha_256(uint8_t *input, uint32_t in_length, uint8_t output[32]);
<b>Function descriptions</b>	calculate digest using SHA256 in HASH mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>input</b>	pointer to the input buffer

Input parameter{in}	
<b>in_length</b>	length of the input buffer
Output parameter{out}	
<b>output</b>	the result digest
Return value	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA256 in HASH mode */
```

```
ErrStatus status = hau_hash_sha_256 (&input, 0x10, output[0]);
```

### hau\_hmac\_sha\_256

The description of hau\_hmac\_sha\_256 is shown as below:

**Table 3-349. Function hau\_hmac\_sha\_256**

<b>Function name</b>	hau_hmac_sha_256
<b>Function prototype</b>	ErrStatus hau_hmac_sha_256(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[32]);
<b>Function descriptions</b>	calculate digest using SHA256 in HMAC mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>key</b>	pointer to the key used for HMAC
Input parameter{in}	
<b>keysize</b>	length of the key used for HMAC
Input parameter{in}	
<b>input</b>	pointer to the input buffer
Input parameter{in}	
<b>in_length</b>	length of the input buffer
Output parameter{out}	
<b>output</b>	the result digest
Return value	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA256 in HMAC mode */
```

```
ErrStatus status = hau_hmac_sha_256 (&key, 0x10, &input, 0x10, output[0]);
```

### hau\_hash\_md5

The description of hau\_hash\_md5 is shown as below:

**Table 3-350. Function hau\_hash\_md5**

<b>Function name</b>	hau_hash_md5
<b>Function prototype</b>	ErrStatus hau_hash_md5(uint8_t *input, uint32_t in_length, uint8_t output[16]);
<b>Function descriptions</b>	calculate digest using MD5 in HASH mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>input</b>	pointer to the input buffer
<b>Input parameter{in}</b>	
<b>in_length</b>	length of the input buffer
<b>Output parameter{out}</b>	
<b>output</b>	the result digest
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* calculate digest using MD5 in HASH mode */
```

```
ErrStatus status = hau_hash_md5 (&input, 0x10, output[0]);
```

## hau\_hmac\_md5

The description of hau\_hmac\_md5 is shown as below:

**Table 3-351. Function hau\_hmac\_md5**

<b>Function name</b>	hau_hmac_md5
<b>Function prototype</b>	ErrStatus hau_hmac_md5(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[16]);
<b>Function descriptions</b>	calculate digest using MD5 in HMAC mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>key</b>	pointer to the key used for HMAC
<b>Input parameter{in}</b>	
<b>keysize</b>	length of the key used for HMAC
<b>Input parameter{in}</b>	
<b>input</b>	pointer to the input buffer
<b>Input parameter{in}</b>	
<b>in_length</b>	length of the input buffer
<b>Output parameter{out}</b>	
<b>output</b>	the result digest
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* calculate digest using MD5 in HMAC mode */
```

```
ErrStatus status = hau_hmac_md5 (&key, 0x10, &input, 0x10, output[0]);
```

## hau\_flag\_get

The description of hau\_flag\_get is shown as below:

**Table 3-352. Function hau\_flag\_get**

<b>Function name</b>	hau_flag_get
<b>Function prototype</b>	FlagStatus hau_flag_get(uint32_t flag);
<b>Function descriptions</b>	get the HAU flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	HAU flag status
HAU_FLAG_DATA_INP UT	there is enough space (16 bytes) in the input FIFO
HAU_FLAG_CALCULA TION_COMPLETE	digest calculation is completed
HAU_FLAG_DMA	DMA is enabled (DMAE =1) or a transfer is processing
HAU_FLAG_BUSY	data block is in process
HAU_FLAG_INFIFO_N O_EMPTY	the input FIFO is not empty
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the HAU flag status */
```

```
FlagStatus status;
```

```
status = hau_flag_get (HAU_FLAG_DMA);
```

## hau\_flag\_clear

The description of hau\_flag\_clear is shown as below:

**Table 3-353. Function hau\_flag\_clear**

<b>Function name</b>	hau_flag_clear
<b>Function prototype</b>	void hau_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear the HAU flag status
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	HAU flag status
<i>HAU_FLAG_DATA_INPUT</i>	there is enough space (16 bytes) in the input FIFO
<i>HAU_FLAG_CALCULATION_COMPLETE</i>	digest calculation is completed
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the HAU flag status */
```

```
hau_flag_clear (HAU_FLAG_DATA_INPUT);
```

## hau\_interrupt\_enable

The description of hau\_interrupt\_enable is shown as below:

**Table 3-354. Function hau\_interrupt\_enable**

<b>Function name</b>	hau_interrupt_enable
<b>Function prototype</b>	void hau_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable the HAU interrupts
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	HAU flag status
<i>HAU_INT_DATA_INPUT</i>	a new block can be entered into the IN buffer
<i>HAU_INT_CALCULATION_COMPLETE</i>	calculation complete
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable hau interrupt */
```

```
hau_interrupt_enable(HAU_INT_DATA_INPUT);
```

## hau\_interrupt\_disable

The description of hau\_interrupt\_disable is shown as below:

**Table 3-355. Function hau\_interrupt\_disable**

<b>Function name</b>	hau_interrupt_disable
<b>Function prototype</b>	void hau_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable the HAU interrupts
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	HAU flag status
HAU_INT_DATA_INPUT	a new block can be entered into the IN buffer
HAU_INT_CALCULATION_COMPLETE	calculation complete
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable hau interrupt */
hau_interrupt_disable(HAU_INT_DATA_INPUT);
```

## hau\_interrupt\_flag\_get

The description of hau\_interrupt\_flag\_get is shown as below:

**Table 3-356. Function hau\_interrupt\_flag\_get**

<b>Function name</b>	hau_interrupt_flag_get
<b>Function prototype</b>	FlagStatus hau_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	get the HAU interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	HAU interrupt flag status
HAU_INT_FLAG_DATA_INPUT	there is enough space (16 bytes) in the input FIFO
HAU_INT_FLAG_CALCULATION_COMPLETE	digest calculation is completed
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	



<b>FlagStatus</b>	SET or RESET
-------------------	--------------

Example:

```
/* get the HAU interrupt flag status */
```

```
FlagStatus status = hau_interrupt_flag_get(HAU_INT_FLAG_DATA_INPUT);
```

## **hau\_interrupt\_flag\_clear**

The description of hau\_interrupt\_flag\_clear is shown as below:

**Table 3-357. Function hau\_interrupt\_flag\_clear**

<b>Function name</b>	hau_interrupt_flag_clear
<b>Function prototype</b>	void hau_interrupt_flag_clear(uint32_t int_flag)
<b>Function descriptions</b>	clear the HAU interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	HAU interrupt flag status
<i>HAU_INT_FLAG_DATA_INPUT</i>	there is enough space (16 bytes) in the input FIFO
<i>HAU_INT_FLAG_CALCULATION_COMPLETE</i>	digest calculation is completed
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the HAU interrupt flag status */
```

```
hau_interrupt_flag_clear(HAU_INT_FLAG_DATA_INPUT);
```

## 3.15. HPDF

A high performance digital filter module (HPDF) for external sigma delta ( $\Sigma$ - $\Delta$ ) modulator is integrated in GD32W51x. The HPDF registers are listed in chapter [3.15.1](#), the HPDF firmware functions are introduced in chapter [3.15.2](#).

### 3.15.1. Descriptions of Peripheral registers

HPDF registers are listed in the table shown as below:

**Table 3-358. HPDF Registers**

Registers	Descriptions
HPDF_CHxCTL	HPDF Channel x control register 0
HPDF_CHxCFG0	HPDF Channel x configuration register 0
HPDF_CHxCFG1	HPDF Channel x configuration register 1
HPDF_CHxTMFDT	HPDF Channel x threshold monitor filter data register
HPDF_CHxPDI	HPDF Channel x input data register
HPDF_CHxPS	HPDF Channel x pulse skip register
HPDF_FLTyCTL0	HPDF Filter y control register 0
HPDF_FLTyCTL1	HPDF Filter y control register 1
HPDF_FLTySTAT	HPDF Filter y status register
HPDF_FLTyINTC	HPDF Filter y interrupt flag clear register
HPDF_FLTyIGCS	HPDF Filter y inserted group channel selection register
HPDF_FLTySFCFG	HPDF Filter y sinc filter configuration register
HPDF_FLTyIDATA	HPDF Filter y inserted group conversion data register
HPDF_FLTyRDATA	HPDF Filter y regular channel conversion data register
HPDF_FLTyTMHT	HPDF Filter y threshold monitor high threshold register
HPDF_FLTyTMLT	HPDF Filter y threshold monitor low threshold register
HPDF_FLTyTMSTAT	HPDF Filter y threshold monitor status register
HPDF_FLTyTMFC	HPDF Filter y threshold monitor flag clear register
HPDF_FLTyEMMAX	HPDF Filter y extremes monitor maximum register
HPDF_FLTyEMMIN	HPDF Filter y extremes monitor minimum register

### 3.15.2. Descriptions of Peripheral functions

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

**Table 3-359. HPDF firmware function**

Function name	Function description
hpdf_deinit	reset HPDF
hpdf_channel_struct_para_init	initialize the parameters of HPDF channel struct with the default values
hpdf_filter_struct_para_init	initialize the parameters of HPDF filter struct with the

Function name	Function description
	default values
hpdf_rc_struct_para_init	initialize the parameters of regular conversion struct with the default values
hpdf_ic_struct_para_init	initialize the parameters of inserted conversion struct with the default values
hpdf_enable	enable the HPDF module globally
hpdf_disable	disable the HPDF module globally
hpdf_channel_init	initialize the HPDF channel
hpdf_filter_init	initialize the HPDF filter
hpdf_rc_init	initialize the regular conversion
hpdf_ic_init	initialize the inserted conversion
hpdf_clock_output_config	configure serial output clock
hpdf_clock_output_source_config	configure serial clock output source
hpdf_clock_output_duty_mode_disable	disable serial clock output duty mode
hpdf_clock_output_duty_mode_enable	enable serial clock output duty mode
hpdf_clock_output_divider_config	configure serial clock output divider
hpdf_channel_enable	enable channel
hpdf_channel_disable	disable channel
hpdf_spi_clock_source_config	configure SPI clock source
hpdf_serial_interface_type_config	configure serial interface type
hpdf_malfunction_monitor_disable	disable malfunction monitor
hpdf_malfunction_monitor_enable	enable malfunction monitor
hpdf_clock_loss_disable	disable clock loss detector
hpdf_clock_loss_enable	enable clock loss detector
hpdf_channel_pin_redirection_disable	disable channel inputs pins redirection
hpdf_channel_pin_redirection_enable	enable channel inputs pins redirection
hpdf_channel_mux_config	configure channel multiplexer select input data source
hpdf_data_pack_mode_config	configure data packing mode
hpdf_data_right_bit_shift_config	configure data right bit-shift
hpdf_calibration_offset_config	configure calibration offset
hpdf_malfunction_break_signal_config	configure malfunction monitor break signal
hpdf_malfunction_counter_config	configure malfunction monitor counter threshold
hpdf_write_parallel_data_standard_mode	write the parallel data on standard mode of data packing
hpdf_write_parallel_data_interleaved_mode	write the parallel data on interleaved mode of data packing
hpdf_write_parallel_data_dual_mode	write the parallel data on dual mode of data packing
hpdf_pulse_skip_update	update the number of pulses to skip
hpdf_pulse_skip_read	read the number of pulses to skip
hpdf_filter_enable	enable filter
hpdf_filter_disable	disable filter
hpdf_filter_config	configure sinc filter order and oversample

## GD32W51x\_F5HC Firmware Library User Guide

Function name	Function description
hpdf_integrator_oversample	configure integrator oversampling rate
hpdf_threshold_monitor_filter_config	configure threshold monitor filter order and oversample
hpdf_threshold_monitor_filter_read_data	read the threshold monitor filter data
hpdf_threshold_monitor_fast_mode_disable	disable threshold monitor fast mode
hpdf_threshold_monitor_fast_mode_enable	enable threshold monitor fast mode
hpdf_threshold_monitor_channel	configure threshold monitor channel
hpdf_threshold_monitor_high_threshold	configure threshold monitor high threshold value
hpdf_threshold_monitor_low_threshold	configure threshold monitor low threshold value
hpdf_high_threshold_break_signal	configure threshold monitor high threshold event break signal
hpdf_low_threshold_break_signal	configure threshold monitor low threshold event break signal
hpdf_extremes_monitor_channel	configure extremes monitor channel
hpdf_extremes_monitor_maximum_get	get the extremes monitor maximum value
hpdf_extremes_monitor_minimum_get	get the extremes monitor minimum value
hpdf_rc_continuous_disable	disable regular conversions continuous mode
hpdf_rc_continuous_enable	enable regular conversions continuous mode
hpdf_rc_start_by_software	start regular channel conversion by software
hpdf_rc_syn_disable	disable regular conversion synchronously
hpdf_rc_syn_enable	enable regular conversion synchronously
hpdf_rc_dma_disable	disable regular conversion DMA channel
hpdf_rc_dma_enable	enable regular conversion DMA channel
hpdf_rc_channel_config	configure regular conversion channel
hpdf_rc_fast_mode_disable	disable regular conversion fast conversion mode
hpdf_rc_fast_mode_enable	enable regular conversion fast conversion mode
hpdf_rc_data_get	get the regular conversion data
hpdf_rc_channel_get	get the channel of regular channel most recently converted
hpdf_ic_start_by_software	start inserted channel conversion by software
hpdf_ic_syn_disable	disable inserted conversion synchronously
hpdf_ic_syn_enable	enable inserted conversion synchronously
hpdf_ic_dma_disable	disable inserted conversion DMA channel
hpdf_ic_dma_enable	enable inserted conversion DMA channel
hpdf_ic_scan_mode_disable	disable scan conversion mode
hpdf_ic_scan_mode_enable	enable scan conversion mode
hpdf_ic_trigger_signal_disable	disable inserted conversions trigger signal
hpdf_ic_trigger_signal_config	configure inserted conversions trigger signal and trigger edge
hpdf_ic_channel_config	configure inserted group conversions channel

Function name	Function description
hpdf_ic_data_get	get the inserted conversions data
hpdf_ic_channel_get	get the channel of inserted group channel most recently converted
hpdf_flag_get	get the HPDF flags
hpdf_flag_clear	clear the HPDF flags
hpdf_interrupt_enable	enable HPDF interrupt
hpdf_interrupt_disable	disable HPDF interrupt
hpdf_interrupt_flag_get	get the HPDF interrupt flags
hpdf_interrupt_flag_clear	clear the HPDF interrupt flags

### Enum hpdf\_channel\_enum

Table 3-360. Enum hpdf\_channel\_enum

enum name	enum description
CHANNEL0	HPDF channel0
CHANNEL1	HPDF channel1

### Enum hpdf\_filter\_enum

Table 3-361. Enum hpdf\_filter\_enum

enum name	enum description
FLT0	HPDF filter0
FLT0	HPDF filter1

### Enum hpdf\_flag\_enum

Table 3-362. Enum hpdf\_flag\_enum

enum name	enum description
HPDF_FLAG_FLTy_ICEF	inserted conversion end flag
HPDF_FLAG_FLTy_RCEF	regular conversion end flag
HPDF_FLAG_FLTy_ICDOF	inserted conversion overflow flag
HPDF_FLAG_FLTy_RCDOF	regular conversion overflow flag
HPDF_FLAG_FLTy_TMEOF	threshold monitor event occurred flag
HPDF_FLAG_FLTy_ICPF	inserted conversion in progress flag
HPDF_FLAG_FLTy_RCPF	regular conversion in progress flag
HPDF_FLAG_FLT0	clock loss on channel 0 flag

enum name	enum description
_CKLF0	
HPDF_FLAG_FLT0 _CKLF1	clock loss on channel 1 flag
HPDF_FLAG_FLT0 _MMF0	malfunction event occurred on channel 0 flag
HPDF_FLAG_FLT0 _MMF1	malfunction event occurred on channel 1 flag
HPDF_FLAG_FLTy _RCHPDT	regular channel pending data
HPDF_FLAG_FLTy _LTF0	threshold monitor low threshold on channel 0 flag
HPDF_FLAG_FLTy _LTF1	threshold monitor low threshold on channel 1 flag
HPDF_FLAG_FLTy _HTF0	threshold monitor high threshold on channel 0 flag
HPDF_FLAG_FLTy _HTF1	threshold monitor high threshold on channel 1 flag

### Enum hpdf\_interrput\_flag\_enum

**Table 3-363. Enum hpdf\_interrput\_flag\_enum**

enum name	enum description
HPDF_INT_FLAG_ FLTy_ICEF	inserted conversion end interrupt flag
HPDF_INT_FLAG_ FLTy_RCEF	regular conversion end interruptflag
HPDF_INT_FLAG_ FLTy_ICDOF	inserted conversion overflow interrupt flag
HPDF_INT_FLAG_ FLTy_RCDOF	regular conversion overflow interrupt flag
HPDF_INT_FLAG_ FLTy_TMEOF	threshold monitor event occurred interrupt flag
HPDF_INT_FLAG_ FLT0_CKLF0	clock loss on channel 0 interrupt flag
HPDF_INT_FLAG_ FLT0_CKLF1	clock loss on channel 1 interrupt flag
HPDF_INT_FLAG_ FLT0_MMFO	malfunction monitor detection on channel 0 interrupt flag
HPDF_INT_FLAG_ FLT0_MMFI	malfunction monitor detection on channel 1 interrupt flag

## Enum hpdf\_interrput\_enum

**Table 3-364. Enum hpdf\_interrput\_enum**

enum name	enum description
HPDF_INT_FLTy_I CEIE	inserted conversion end interrupt enable
HPDF_INT_FLTy_R CEIE	regular conversion end interrupt enable
HPDF_INT_FLTy_I CDOIE	inserted conversion data overflow interrupt enable
HPDF_INT_FLTy_R CDOIE	regular conversion data overflow interrupt enable
HPDF_INT_FLTy_T MIE	threshold monitor interrupt enable
HPDF_INT_FLT0_M MIE	malfunction monitor interrupt enable
HPDF_INT_FLT0_C KLIE	clock loss interrupt enable

## Structure hpdf\_channel\_parameter\_struct

**Table 3-365. Structure hpdf\_channel\_parameter\_struct**

Member name	Function description
data_packing_mode	ata packing mode for HPDF_CHxPDI register
channel_muxlexer	channel multiplexer select input data source
channel_pin_select	channel inputs pins selection
ck_loss_detector	clock loss detector
malfunction_monitor	malfunction monitor
spi_ck_source	SPI clock source select
serial_interface	serial interface type
calibration_offset	24-bit calibration offset
right_bit_shift	data right bit-shift
tm_filter	threshold monitor Sinc filter order selection
tm_filter_oversample	threshold monitor filter oversampling rate
mm_break_signal	malfunction monitor break signal distribution
mm_counter_threshold	malfunction monitor counter threshold
plsk_value	the number of serial input samples that will be skipped

## Structure hpdf\_filter\_parameter\_struct

**Table 3-366. Structure hpdf\_filter\_parameter\_struct**

Member name	Function description
tm_fast_mode	threshold monitor fast mode
tm_channel	threshold monitor channel

tm_high_threshold	threshold monitor high threshold
tm_low_threshold	threshold monitor low threshold value
extreme_monitor_channel	extremes monitor channel
sinc_filter	sinc filter order
sinc_oversample	sinc filter oversampling rate
integrator_oversample	integrator oversampling rate
ht_break_signal	high threshold event break signal distribution
lt_break_signal	low threshold event break signal distribution

### Structure hpdf\_rc\_parameter\_struct

**Table 3-367. Structure hpdf\_rc\_parameter\_struct**

Member name	Function description
fast_mode	fast conversion mode enable for regular conversions
rcs_channel	regular conversion channel
rcdmaen	DMA channel enabled to read data for the regular conversion
rcsyn	regular conversion synchronously
continuous_mode	regular conversions continuous mode

### Structure hpdf\_ic\_parameter\_struct

**Table 3-368. Structure hpdf\_ic\_parameter\_struct**

Member name	Function description
trigger_dege	inserted conversions trigger edge
trigger_signal	inserted conversions trigger signal
icdmaen	DMA channel enabled to read data for the inserted channel group
scmod	scan conversion mode of inserted conversions
icsyn	inserted conversion synchronously
ic_channel_group	inserted channel group selection

### hpdf\_deinit

The description of hpdf\_deinit is shown as below:

**Table 3-369. Function hpdf\_deinit**

<b>Function name</b>	hpdf_deinit
<b>Function prototype</b>	void hpdf_deinit(void);
<b>Function descriptions</b>	reset HPDF
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-



Return value	
-	-

Example:

```
/* reset HPDF */
```

```
hpdf_deinit();
```

## hpdf\_channel\_struct\_para\_init

The description of hpdf\_channel\_struct\_para\_init is shown as below:

**Table 3-370. Function hpdf\_channel\_struct\_para\_init**

Function name	hpdf_channel_struct_para_init
Function prototype	void hpdf_channel_struct_para_init(hpdf_channel_parameter_struct* init_struct);
Function descriptions	initialize the parameters of HPDF channel struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
*init_struct	the initialized struct needed to initialize HPDF channel, refer to <a href="#">Table 3-365. Structure hpdf_channel_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the parameters of HPDF channel */
```

```
hpdf_channel_parameter_struct hpdf_channel_init_struct;
```

```
hpdf_channel_struct_para_init(&hpdf_channel_init_struct);
```

## hpdf\_filter\_struct\_para\_init

The description of hpdf\_filter\_struct\_para\_init is shown as below:

**Table 3-371. Function hpdf\_filter\_struct\_para\_init**

Function name	hpdf_filter_struct_para_init
Function prototype	void hpdf_filter_struct_para_init(hpdf_filter_parameter_struct* init_struct);
Function descriptions	initialize the parameters of HPDF filter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
*init_struct	the initialized struct needed to initialize HPDF filter, refer to <a href="#">Table 3-366.</a>

	<a href="#">Structure hpdf_filter_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the parameters of HPDF filter */
```

```
hpdf_filter_parameter_struct hpdf_filter_init_struct;
```

```
hpdf_filter_struct_para_init(&hpdf_filter_init_struct);
```

### hpdf\_rc\_struct\_para\_init

The description of hpdf\_rc\_struct\_para\_init is shown as below:

**Table 3-372. Function hpdf\_rc\_struct\_para\_init**

<b>Function name</b>	hpdf_rc_struct_para_init
<b>Function prototype</b>	void hpdf_rc_struct_para_init(hpdf_rc_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize the parameters of regular conversion struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>*init_struct</b>	the initialized struct needed to initialize HPDF regular conversion, refer to <a href="#">Table 3-367. Structure hpdf_rc_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the parameters of regular conversion */
```

```
hpdf_rc_parameter_struct hpdf_rc_init_struct;
```

```
hpdf_rc_struct_para_init(&hpdf_rc_init_struct);
```

### hpdf\_ic\_struct\_para\_init

The description of hpdf\_ic\_struct\_para\_init is shown as below:

**Table 3-373. Function hpdf\_ic\_struct\_para\_init**

<b>Function name</b>	hpdf_ic_struct_para_init
<b>Function prototype</b>	void hpdf_ic_struct_para_init(hpdf_ic_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize the parameters of inserted conversion struct with the default values
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>*init_struct</b>	the initialized struct needed to initialize HPDF inserted conversion, refer to <a href="#">Table 3-368. Structure hpdf_ic_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the parameters of inserted conversion */
hpdf_ic_parameter_struct hpdf_ic_init_struct;
hpdf_ic_struct_para_init(&hpdf_ic_init_struct);
```

### hpdf\_enable

The description of hpdf\_enable is shown as below:

**Table 3-374. Function hpdf\_enable**

<b>Function name</b>	hpdf_enable
<b>Function prototype</b>	void hpdf_enable(void);
<b>Function descriptions</b>	enable the HPDF module globally
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the HPDF module */
hpdf_enable();
```

### hpdf\_disable

The description of hpdf\_disable is shown as below:

**Table 3-375. Function hpdf\_disable**

<b>Function name</b>	hpdf_disable
<b>Function prototype</b>	void hpdf_disable(void);
<b>Function descriptions</b>	disable the HPDF module globally

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the HPDF module */
```

```
hpdf_disable();
```

### hpdf\_channel\_init

The description of hpdf\_channel\_init is shown as below:

**Table 3-376. Function hpdf\_channel\_init**

<b>Function name</b>	hpdf_channel_init
<b>Function prototype</b>	void hpdf_channel_init(hpdf_channel_enum channelx, hpdf_channel_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize the HPDF channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Input parameter{in}</b>	
<b>*init_struct</b>	the initialized struct needed to initialize HPDF channel, refer to <a href="#">Table 3-365. Structure hpdf_channel_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the HPDF channel0 */
```

```
hpdf_channel_parameter_struct hpdf_channel_init_struct;
```

```
/* initialize HPDF channel0 */
```

```
hpdf_channel_init_struct.data_packing_mode = DPM_STANDARD_MODE;
```

```
hpdf_channel_init_struct.malfunction_monitor = MM_ENABLE;
```

```

hpdf_channel_init_struct.spi_ck_source = EXTERNAL_CKIN;

hpdf_channel_init_struct.channel_muxlexer = SERIAL_INPUT;

hpdf_channel_init_struct.serial_interface = SPI_RISING_EDGE;

hpdf_channel_init_struct.calibration_offset = 0;

hpdf_channel_init_struct.right_bit_shift = 0;

hpdf_channel_init_struct.mm_counter_threshold = 110;

hpdf_channel_init_struct.plsk_value = 0;

hpdf_channel_init(CHANNEL0, &hpdf_channel_init_struct);

```

### hpdf\_filter\_init

The description of hpdf\_filter\_init is shown as below:

**Table 3-377. Function hpdf\_filter\_init**

<b>Function name</b>	hpdf_filter_init
<b>Function prototype</b>	void hpdf_filter_init(hpdf_filter_enum filtery, hpdf_filter_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize the HPDF filter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Input parameter{in}</b>	
<b>*init_struct</b>	the initialized struct needed to initialize HPDF filter, refer to <a href="#">Table 3-366. Structure hpdf_filter_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* initialize the HPDF fliter0 */

hpdf_filter_parameter_struct hpdf_filter_init_struct;

hpdf_filter_init_struct.tm_fast_mode = TMFM_DISABLE;

hpdf_filter_init_struct.tm_channel = TMCHEN_CHANNEL0;

hpdf_filter_init_struct.tm_high_threshold = tm_high_val;

hpdf_filter_init_struct.tm_low_threshold = tm_low_val;

```

```

hpdf_filter_init_struct.extreme_monitor_channel = EM_CHANNEL0;

hpdf_filter_init_struct.sinc_filter            = FLT_SINC3;

hpdf_filter_init_struct.sinc_oversample       = FLT_OVER_SAMPLE_32;

hpdf_filter_init_struct.integrator_oversample = INTEGRATOR_BYPASS;

hpdf_filter_init(FLT0, &hpdf_filter_init_struct);

```

## hpdf\_rc\_init

The description of hpdf\_rc\_init is shown as below:

**Table 3-378. Function hpdf\_rc\_init**

Function name	hpdf_rc_init
Function prototype	void hpdf_rc_init(hpdf_filter_enum filtery, hpdf_rc_parameter_struct* init_struct);
Function descriptions	initialize the regular conversion
Precondition	-
The called functions	-
Input parameter{in}	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
Input parameter{in}	
<b>*init_struct</b>	the initialized struct needed to initialize HPDF regular conversion, refer to <a href="#">Table 3-367. Structure hpdf_rc_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize regular conversion of the HPDF filter0 */

hpdf_rc_parameter_struct hpdf_rc_init_struct;

hpdf_rc_init_struct.fast_mode = FAST_DISABLE;

hpdf_rc_init_struct.rcs_channel = RCS_CHANNEL0;

hpdf_rc_init_struct.rcdmaen = RCDMAEN_ENABLE;

hpdf_rc_init_struct.continuous_mode = RCCM_ENABLE;

hpdf_rc_init(FLT0, &hpdf_rc_init_struct);

```

## hpdf\_ic\_init

The description of hpdf\_ic\_init is shown as below:

Table 3-379. Function `hpdf_ic_init`

Function name	<code>hpdf_ic_init</code>
Function prototype	<code>void hpdf_ic_init(hpdf_filter_enum filtery, hpdf_ic_parameter_struct* init_struct);</code>
Function descriptions	initialize the inserted conversion
Precondition	-
The called functions	-
Input parameter{in}	
<code>filtery</code>	The filter of HPDF module
<code>FLTy(y=0..1)</code>	select HPDF filter, refer to <a href="#">Table 3-361. Enum <code>hpdf_filter_enum</code></a>
Input parameter{in}	
<code>*init_struct</code>	the initialized struct needed to initialize HPDF inserted conversion, refer to <a href="#">Table 3-368. Structure <code>hpdf_ic_parameter_struct</code></a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize inserted conversion of the HPDF filter0 */

hpdf_ic_parameter_struct hpdf_ic_init_struct;

hpdf_ic_init_struct.ic_channel_group = IGCSEL_CHANNEL0_1;

hpdf_ic_init_struct.scmmod = SCMOD_ENABLE;

hpdf_ic_init_struct.icdmaen = ICDMAEN_ENABLE;

hpdf_ic_init_struct.icsyn = ICSYN_DISABLE ;

hpdf_ic_init(FLT0, &hpdf_ic_init_struct);

```

### `hpdf_clock_output_config`

The description of `hpdf_clock_output_config` is shown as below:

Table 3-380. Function `hpdf_clock_output_config`

Function name	<code>hpdf_clock_output_config</code>
Function prototype	<code>void hpdf_clock_output_config(uint32_t source, uint8_t divider, uint32_t mode);</code>
Function descriptions	configure serial output clock
Precondition	disable HPDF
The called functions	-
Input parameter{in}	
<code>source</code>	the HPDF serial clock output source
<code>SERIAL_SYSTEM_CLK</code>	serial clock output source is from system clock

<i>SERIAL_SYSTEM_CLK</i>	serial clock output source is from audio clock
<b>Input parameter{in}</b>	
<b>divider</b>	serial clock output divider(0-255)
<b>Input parameter{in}</b>	
<b>mode</b>	serial clock output duty mode
<i>CKOUTDM_DISABLE</i>	disable serial clock output duty mode
<i>CKOUTDM_ENABLE</i>	enable serial clock output duty mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure serial clock output */
```

```
hpdf_clock_output_config(SERIAL_SYSTEM_CLK, 175, CKOUTDM_ENABLE);
```

## hpdf\_clock\_output\_source\_config

The description of hpdf\_clock\_output\_source\_config is shown as below:

**Table 3-381. Function hpdf\_clock\_output\_source\_config**

<b>Function name</b>	hpdf_clock_output_source_config
<b>Function prototype</b>	void hpdf_clock_output_source_config(uint32_t source);
<b>Function descriptions</b>	configure serial clock output source
<b>Precondition</b>	disable HPDF
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	the HPDF serial clock output source
<i>SERIAL_SYSTEM_CLK</i>	serial clock output source is from system clock
<i>SERIAL_SYSTEM_CLK</i>	serial clock output source is from audio clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure serial clock output source */
```

```
hpdf_clock_output_config(SERIAL_SYSTEM_CLK);
```

## hpdf\_clock\_output\_duty\_mode\_disable

The description of hpdf\_clock\_output\_duty\_mode\_disable is shown as below:



**Table 3-382. Function hpdf\_clock\_output\_duty\_mode\_disable**

<b>Function name</b>	hpdf_clock_output_duty_mode_disable
<b>Function prototype</b>	void hpdf_clock_output_duty_mode_disable(void);
<b>Function descriptions</b>	disable serial clock output duty mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable serial clock output duty mode*/
```

```
hpdf_clock_output_duty_mode_disable();
```

### hpdf\_clock\_output\_duty\_mode\_enable

The description of hpdf\_clock\_output\_duty\_mode\_enable is shown as below:

**Table 3-383. Function hpdf\_clock\_output\_duty\_mode\_enable**

<b>Function name</b>	hpdf_clock_output_duty_mode_enable
<b>Function prototype</b>	void hpdf_clock_output_duty_mode_enable(void);
<b>Function descriptions</b>	enable serial clock output duty mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable serial clock output duty mode*/
```

```
hpdf_clock_output_duty_mode_enable();
```

### hpdf\_clock\_output\_divider\_config

The description of hpdf\_clock\_output\_divider\_config is shown as below:

**Table 3-384. Function hpdf\_clock\_output\_divider\_config**

<b>Function name</b>	hpdf_clock_output_divider_config
----------------------	----------------------------------

<b>Function prototype</b>	void hpdf_clock_output_divider_config(uint8_t divider);
<b>Function descriptions</b>	configure serial clock output divider
<b>Precondition</b>	disable HPDF
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>divider</b>	serial clock output divider(0-255)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure serial clock output divider */
hpdf_clock_output_divider_config (255);
```

## hpdf\_channel\_enable

The description of hpdf\_channel\_enable is shown as below:

**Table 3-385. Function hpdf\_channel\_enable**

<b>Function name</b>	hpdf_channel_enable
<b>Function prototype</b>	void hpdf_channel_enable(hpdf_channel_enum channelx);
<b>Function descriptions</b>	enable channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable channel0 */
hpdf_channel_enable(CHANNEL0);
```

## hpdf\_channel\_disable

The description of hpdf\_channel\_disable is shown as below:

**Table 3-386. Function hpdf\_channel\_disable**

<b>Function name</b>	hpdf_channel_disable
<b>Function prototype</b>	void hpdf_channel_disable(hpdf_channel_enum channelx);

<b>Function descriptions</b>	disable channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable channel0 */
```

```
hpdf_channel_disable(CHANNEL0);
```

## hpdf\_spi\_clock\_source\_config

The description of hpdf\_spi\_clock\_source\_config is shown as below:

**Table 3-387. Function hpdf\_spi\_clock\_source\_config**

<b>Function name</b>	hpdf_spi_clock_source_config
<b>Function prototype</b>	void hpdf_spi_clock_source_config(hpdf_channel_enum channelx, uint32_t clock_source);
<b>Function descriptions</b>	configure SPI clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Input parameter{in}</b>	
<b>clock_source</b>	SPI clock source
<i>EXTERNAL_CKIN</i>	external input clock
<i>INTERNAL_CKOUT</i>	internal CKOUT clock
<i>HALF_CKOUT_FALLIN_G_EDGE</i>	internal CKOUT clock, sampling point on each second CKOUT falling edge
<i>HALF_CKOUT_RISING_EDGE</i>	internal CKOUT clock, sampling point on each second CKOUT rising edge
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure external input clock as SPI clock source */
```

```
hpdf_spi_clock_source_config(CHANNEL0, EXTERNAL_CKIN);
```

### hpdf\_serial\_interface\_type\_config

The description of hpdf\_serial\_interface\_type\_config is shown as below:

**Table 3-388. Function hpdf\_serial\_interface\_type\_config**

<b>Function name</b>	hpdf_serial_interface_type_config
<b>Function prototype</b>	void hpdf_serial_interface_type_config(hpdf_channel_enum channelx, uint32_t type);
<b>Function descriptions</b>	configure serial interface type
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
CHANNELx(x=0..1)	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Input parameter{in}</b>	
<b>type</b>	serial interface type
SPI_RISING_EDGE	SPI interface, sample data on rising edge
SPI_FALLING_EDGE	SPI interface, sample data on rising edge
MANCHESTER_CODE 0	Manchester coded input: rising edge = logic 0, falling edge = logic 1
MANCHESTER_CODE 1	Manchester coded input: rising edge = logic 1, falling edge = logic 0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure external input clock as SPI clock source */
```

```
hpdf_spi_clock_source_config(CHANNEL0, EXTERNAL_CKIN);
```

### hpdf\_malfunction\_monitor\_disable

The description of hpdf\_malfunction\_monitor\_disable is shown as below:

**Table 3-389. Function hpdf\_malfunction\_monitor\_disable**

<b>Function name</b>	hpdf_malfunction_monitor_disable
<b>Function prototype</b>	void hpdf_malfunction_monitor_disable(hpdf_channel_enum channelx);
<b>Function descriptions</b>	disable malfunction monitor
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable malfunction monitor */
```

```
hpdf_malfunction_monitor_disable(CHANNEL0);
```

## hpdf\_malfunction\_monitor\_enable

The description of hpdf\_malfunction\_monitor\_enable is shown as below:

**Table 3-390. Function hpdf\_malfunction\_monitor\_enable**

<b>Function name</b>	hpdf_malfunction_monitor_enable
<b>Function prototype</b>	void hpdf_malfunction_monitor_enable(hpdf_channel_enum channelx);
<b>Function descriptions</b>	enable malfunction monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable malfunction monitor */
```

```
hpdf_malfunction_monitor_enable(CHANNEL1);
```

## hpdf\_clock\_loss\_disable

The description of hpdf\_clock\_loss\_disable is shown as below:

**Table 3-391. Function hpdf\_clock\_loss\_disable**

<b>Function name</b>	hpdf_clock_loss_disable
<b>Function prototype</b>	void hpdf_clock_loss_disable(hpdf_channel_enum channelx);
<b>Function descriptions</b>	disable clock loss detector
<b>Precondition</b>	disable CHANNELx(x=0..1)
<b>The called functions</b>	-

Input parameter{in}	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable clock loss detector */
```

```
hpdf_clock_loss_disable(CHANNEL0);
```

### hpdf\_clock\_loss\_enable

The description of hpdf\_clock\_loss\_enable is shown as below:

**Table 3-392. Function hpdf\_clock\_loss\_enable**

<b>Function name</b>	hpdf_clock_loss_enable
<b>Function prototype</b>	void hpdf_clock_loss_enable(hpdf_channel_enum channelx);
<b>Function descriptions</b>	enable clock loss detector
<b>Precondition</b>	disable CHANNELx(x=0..1)
<b>The called functions</b>	-
Input parameter{in}	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable clock loss detector */
```

```
hpdf_clock_loss_enable(CHANNEL0);
```

### hpdf\_channel\_pin\_redirection\_disable

The description of hpdf\_channel\_pin\_redirection\_disable is shown as below:

**Table 3-393. Function hpdf\_channel\_pin\_redirection\_disable**

<b>Function name</b>	hpdf_channel_pin_redirection_disable
<b>Function prototype</b>	void hpdf_channel_pin_redirection_disable(hpdf_channel_enum channelx);
<b>Function descriptions</b>	disable channel inputs pins redirection
<b>Precondition</b>	disable CHANNELx(x=0..1)
<b>The called functions</b>	-

Input parameter{in}	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable channel0 inputs pins redirection */
```

```
hpdf_channel_pin_redirection_disable(CHANNEL0);
```

## hpdf\_channel\_pin\_redirection\_enable

The description of hpdf\_channel\_pin\_redirection\_enable is shown as below:

**Table 3-394. Function hpdf\_channel\_pin\_redirection\_enable**

<b>Function name</b>	hpdf_channel_pin_redirection_enable
<b>Function prototype</b>	void hpdf_channel_pin_redirection_enable(hpdf_channel_enum channelx);
<b>Function descriptions</b>	enable channel inputs pins redirection
<b>Precondition</b>	disable CHANNELx(x=0..1)
<b>The called functions</b>	-
Input parameter{in}	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable channel0 inputs pins redirection */
```

```
hpdf_channel_pin_redirection_enable(CHANNEL0);
```

## hpdf\_channel\_multiplexer\_config

The description of hpdf\_channel\_multiplexer\_config is shown as below:

**Table 3-395. Function hpdf\_channel\_multiplexer\_config**

<b>Function name</b>	hpdf_channel_multiplexer_config
<b>Function prototype</b>	void hpdf_channel_multiplexer_config(hpdf_channel_enum channelx, uint32_t data_source);
<b>Function descriptions</b>	configure channel multiplexer select input data source
<b>Precondition</b>	disable CHANNELx(x=0..1)

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Input parameter{in}</b>	
<b>data_source</b>	input data source
<i>SERIAL_INPUT</i>	input data source is taken from serial inputs
<i>INTERNAL_INPUT</i>	input data source is taken from internal HPDF_CHxPDI register
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure channel multiplexer select input data source */
```

```
hpdf_channel_mux_config(CHANNEL0, SERIAL_INPUT);
```

## hpdf\_data\_pack\_mode\_config

The description of hpdf\_data\_pack\_mode\_config is shown as below:

**Table 3-396. Function hpdf\_data\_pack\_mode\_config**

<b>Function name</b>	hpdf_data_pack_mode_config
<b>Function prototype</b>	void hpdf_data_pack_mode_config(hpdf_channel_enum channelx, uint32_t mode);
<b>Function descriptions</b>	configure data packing mode
<b>Precondition</b>	disable CHANNELx(x=0..1)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Input parameter{in}</b>	
<b>mode</b>	parallel data packing mode
<i>DPM_STANDARD_MODE</i>	standard mode
<i>DPM_INTERLEAVED_MODE</i>	interleaved mode
<i>DPM_DUAL_MODE</i>	dual mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:



```
/* configure data packing mode */
```

```
hpdf_data_pack_mode_config(CHANNEL0, DPM_STANDARD_MODE);
```

## hpdf\_data\_right\_bit\_shift\_config

The description of hpdf\_data\_right\_bit\_shift\_config is shown as below:

**Table 3-397. Function hpdf\_data\_right\_bit\_shift\_config**

<b>Function name</b>	hpdf_data_right_bit_shift_config
<b>Function prototype</b>	void hpdf_data_right_bit_shift_config(hpdf_channel_enum channelx, uint8_t right_shift);
<b>Function descriptions</b>	configure data right bit-shift
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
CHANNELx(x=0..1)	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Input parameter{in}</b>	
<b>right_shift</b>	the number of bits that determine the right shift(0-31)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure data right bit-shift */
```

```
hpdf_data_right_bit_shift_config(CHANNEL0, 5);
```

## hpdf\_calibration\_offset\_config

The description of hpdf\_calibration\_offset\_config is shown as below:

**Table 3-398. Function hpdf\_calibration\_offset\_config**

<b>Function name</b>	hpdf_calibration_offset_config
<b>Function prototype</b>	void hpdf_calibration_offset_config(hpdf_channel_enum channelx, int32_t offset);
<b>Function descriptions</b>	configure calibration offset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
CHANNELx(x=0..1)	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Input parameter{in}</b>	
<b>offset</b>	24-bit calibration offset, must be in (-8388608~8388607)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure calibration offset */
```

```
hpdf_calibration_offset_config (CHANNEL0, -255);
```

## hpdf\_malfunction\_break\_signal\_config

The description of hpdf\_malfunction\_break\_signal\_config is shown as below:

**Table 3-399. Function hpdf\_malfunction\_break\_signal\_config**

Function name	hpdf_malfunction_break_signal_config
Function prototype	void hpdf_malfunction_break_signal_config(hpdf_channel_enum channelx, uint32_t break_signal);
Function descriptions	configure malfunction monitor break signal
Precondition	-
The called functions	-
Input parameter{in}	
channelx	The channel of HPDF module
CHANNELx(x=0..1)	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
Input parameter{in}	
break_signal	malfunction monitor break signal distribution
NO_MM_BREAK	break signal is not distributed to malfunction monitor on channel
MM_BREAK0	break signal 0 is distributed to malfunction monitor on channel
MM_BREAK1	break signal 1 is distributed to malfunction monitor on channel
MM_BREAK0_1	break signal 0 and 1 is distributed to malfunction monitor on channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure break signal is distributed to malfunction monitor on channel0 */
```

```
hpdf_data_pack_mode_config(CHANNEL0, MM_BREAK0_1);
```

## hpdf\_malfunction\_counter\_config

The description o hpdf\_malfunction\_counter\_config is shown as below:

**Table 3-400. Function hpdf\_malfunction\_counter\_config**

Function name	hpdf_malfunction_counter_config
---------------	---------------------------------

<b>Function prototype</b>	void hpdf_malfunction_counter_config(hpdf_channel_enum channelx, uint8_t threshold);
<b>Function descriptions</b>	configure malfunction monitor counter threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Input parameter{in}</b>	
<b>threshold</b>	malfunction monitor counter threshold(0-255)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure malfunction monitor counter threshold */
hpdf_malfunction_counter_config(CHANNEL0, 255);
```

### hpdf\_write\_parallel\_data\_standard\_mode

The description of hpdf\_write\_parallel\_data\_standard\_mode is shown as below:

**Table 3-401. Function hpdf\_write\_parallel\_data\_standard\_mode**

<b>Function name</b>	hpdf_write_parallel_data_standard_mode
<b>Function prototype</b>	void hpdf_write_parallel_data_standard_mode(hpdf_channel_enum channelx, int16_t data);
<b>Function descriptions</b>	write the parallel data on standard mode of data packing
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Input parameter{in}</b>	
<b>data</b>	the parallel data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* write the parallel data on standard mode of data packing */
hpdf_write_parallel_data_standard_mode(CHANNEL0, 0xEFFF);
```

## hpdf\_write\_parallel\_data\_interleaved\_mode

The description of hpdf\_write\_parallel\_data\_interleaved\_mode is shown as below:

**Table 3-402. Function hpdf\_write\_parallel\_data\_interleaved\_mode**

<b>Function name</b>	hpdf_write_parallel_data_interleaved_mode
<b>Function prototype</b>	void hpdf_write_parallel_data_interleaved_mode(hpdf_channel_enum channelx, int32_t data);
<b>Function descriptions</b>	write the parallel data on interleaved mode of data packing
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Input parameter{in}</b>	
<b>data</b>	the parallel data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* write the parallel data on interleaved mode of data packing */
```

```
hpdf_write_parallel_data_interleaved_mode(CHANNEL0, 0xEFFFFFFF);
```

## hpdf\_write\_parallel\_data\_dual\_mode

The description of hpdf\_write\_parallel\_data\_dual\_mode is shown as below:

**Table 3-403. Function hpdf\_write\_parallel\_data\_dual\_mode**

<b>Function name</b>	hpdf_write_parallel_data_dual_mode
<b>Function prototype</b>	void hpdf_write_parallel_data_dual_mode(hpdf_channel_enum channelx, int32_t data);
<b>Function descriptions</b>	write the parallel data on dual mode of data packing
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Input parameter{in}</b>	
<b>data</b>	the parallel data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* write the parallel data on dual mode of data packing */
```

```
hpdf_write_parallel_data_dual_mode(CHANNEL0, 0xEFFFFFFF);
```

## hpdf\_pulse\_skip\_update

The description of hpdf\_pulse\_skip\_update is shown as below:

**Table 3-404. Function hpdf\_pulse\_skip\_update**

<b>Function name</b>	hpdf_pulse_skip_update
<b>Function prototype</b>	void hpdf_pulse_skip_update(hpdf_channel_enum channelx, uint8_t number);
<b>Function descriptions</b>	update the number of pulses to skip
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Input parameter{in}</b>	
<b>number</b>	the number of serial input samples that will be skipped
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* update the number of pulses to skip */
```

```
pdf_pulse_skip_update(CHANNEL0, 63);
```

## hpdf\_pulse\_skip\_read

The description of hpdf\_pulse\_skip\_read is shown as below:

**Table 3-405. Function hpdf\_pulse\_skip\_read**

<b>Function name</b>	hpdf_pulse_skip_read
<b>Function prototype</b>	uint8_t hpdf_pulse_skip_read(hpdf_channel_enum channelx);
<b>Function descriptions</b>	read the number of pulses to skip
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>

Output parameter{out}	
-	-
Return value	
uint8_t	the number of pulses to skip

Example:

```
/* read the number of pulses to skip */
uint8_t value;
value = hpdf_pulse_skip_read(CHANNEL0);
```

## hpdf\_filter\_enable

The description of hpdf\_filter\_enable is shown as below:

**Table 3-406. Function hpdf\_filter\_enable**

Function name	hpdf_filter_enable
Function prototype	void hpdf_filter_enable(hpdf_filter_enum filtery);
Function descriptions	enable filter
Precondition	-
The called functions	-
Input parameter{in}	
filtery	The filter of HPDF module
FLTy(y=0..1)	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable filter0 */
hpdf_filter_enable(FLT0);
```

## hpdf\_filter\_disable

The description of hpdf\_filter\_disable is shown as below:

**Table 3-407. Function hpdf\_filter\_disable**

Function name	hpdf_filter_disable
Function prototype	void hpdf_filter_disable(hpdf_filter_enum filtery);
Function descriptions	disable filter
Precondition	-
The called functions	-
Input parameter{in}	

<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable filter0 */
```

```
hpdf_filter_disable(FLT0);
```

## hpdf\_filter\_config

The description of hpdf\_filter\_config is shown as below:

**Table 3-408. Function hpdf\_filter\_config**

<b>Function name</b>	hpdf_filter_config
<b>Function prototype</b>	void hpdf_filter_config(hpdf_filter_enum filtery, uint32_t order, uint16_t oversample);
<b>Function descriptions</b>	configure sinc filter order and oversample
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Input parameter{in}</b>	
<b>order</b>	sinc filter order
<i>FLT_FASTSINC</i>	FastSinc filter type
<i>FLT_SINC1</i>	Sinc1 filter type
<i>FLT_SINC2</i>	Sinc2 filter type
<i>FLT_SINC3</i>	Sinc3 filter type
<i>FLT_SINC4</i>	Sinc4 filter type
<i>FLT_SINC5</i>	Sinc5 filter type
<b>Input parameter{in}</b>	
<b>oversample</b>	Sinc filter oversampling rate(1-1024)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure sinc filter order and oversample */
```

```
hpdf_filter_config(FLT0, FLT_SINC2, 64);
```

## hpdf\_integrator\_oversample

The description of hpdf\_integrator\_oversample is shown as below:

**Table 3-409. Function hpdf\_integrator\_oversample**

<b>Function name</b>	hpdf_integrator_oversample
<b>Function prototype</b>	void hpdf_integrator_oversample(hpdf_filter_enum filtery, uint8_t oversample);
<b>Function descriptions</b>	configure integrator oversampling rate
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Input parameter{in}</b>	
<b>oversample</b>	integrator oversampling rate(1-256)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure integrator oversampling rate */
hpdf_integrator_oversample(FLT0, 256);
```

## hpdf\_threshold\_monitor\_filter\_config

The description of hpdf\_threshold\_monitor\_filter\_config is shown as below:

**Table 3-410. Function hpdf\_threshold\_monitor\_filter\_config**

<b>Function name</b>	hpdf_threshold_monitor_filter_config
<b>Function prototype</b>	void hpdf_threshold_monitor_filter_config(hpdf_channel_enum channelx, uint32_t order, uint8_t oversample);
<b>Function descriptions</b>	configure threshold monitor filter order and oversample
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Input parameter{in}</b>	
<b>order</b>	threshold monitor Sinc filter order
<i>TM_FASTSINC</i>	FastSinc filter type
<i>TM_SINC1</i>	Sinc1 filter type
<i>TM_SINC2</i>	Sinc2 filter type



<i>TM_SINC3</i>	Sinc3 filter type
<b>Input parameter{in}</b>	
<b>oversample</b>	Sinc filter oversampling rate(1-32)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure threshold monitor filter order and oversample */
```

```
hpdf_threshold_monitor_filter_config (FLT0, TM_SINC3);
```

### hpdf\_threshold\_monitor\_filter\_read\_data

The description of hpdf\_threshold\_monitor\_filter\_read\_data is shown as below:

**Table 3-411. Function hpdf\_threshold\_monitor\_filter\_read\_data**

<b>Function name</b>	hpdf_threshold_monitor_filter_read_data
<b>Function prototype</b>	int16_t hpdf_threshold_monitor_filter_read_data(hpdf_channel_enum channelx);
<b>Function descriptions</b>	read the threshold monitor filter data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>int16_t</b>	the threshold monitor filter data

Example:

```
/* read the threshold monitor filter data */
```

```
int16_t data;
```

```
data = hpdf_threshold_monitor_filter_read_data(CHANNEL0);
```

### hpdf\_threshold\_monitor\_fast\_mode\_disable

The description of hpdf\_threshold\_monitor\_fast\_mode\_disable is shown as below:

**Table 3-412. Function hpdf\_threshold\_monitor\_fast\_mode\_disable**

<b>Function name</b>	hpdf_threshold_monitor_fast_mode_disable
<b>Function prototype</b>	void hpdf_threshold_monitor_fast_mode_disable(hpdf_filter_enum filtery);

<b>Function descriptions</b>	disable threshold monitor fast mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable threshold monitor fast mode */
hpdf_threshold_monitor_fast_mode_disable (FLT0);
```

## hpdf\_threshold\_monitor\_fast\_mode\_enable

The description of hpdf\_threshold\_monitor\_fast\_mode\_enable is shown as below:

**Table 3-413. Function hpdf\_threshold\_monitor\_fast\_mode\_enable**

<b>Function name</b>	hpdf_threshold_monitor_fast_mode_enable
<b>Function prototype</b>	void hpdf_threshold_monitor_fast_mode_enable(hpdf_filter_enum filtery);
<b>Function descriptions</b>	enable threshold monitor fast mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable threshold monitor fast mode */
hpdf_threshold_monitor_fast_mode_enable(FLT0);
```

## hpdf\_threshold\_monitor\_channel

The description of hpdf\_threshold\_monitor\_channel is shown as below:

**Table 3-414. Function hpdf\_threshold\_monitor\_channel**

<b>Function name</b>	hpdf_threshold_monitor_channel
<b>Function prototype</b>	void hpdf_threshold_monitor_channel(hpdf_filter_enum filtery, uint32_t

	channel);
<b>Function descriptions</b>	configure threshold monitor channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Input parameter{in}</b>	
<b>channel</b>	which channel use threshold monitorx(x=0,1)
<i>TMCHEN_DISABLE</i>	threshold monitorx is disabled on channel 0 and channel 1
<i>TMCHEN_CHANNEL0</i>	threshold monitor x is enabled on channel 0
<i>TMCHEN_CHANNEL1</i>	threshold monitor x is enabled on channel 1
<i>TMCHEN_CHANNEL0_1</i>	threshold monitor x is enabled on channel 0 and channel 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure threshold monitor channel */
```

```
hpdf_threshold_monitor_channel(FLT0, TMCHEN_CHANNEL0_1);
```

## hpdf\_threshold\_monitor\_high\_threshold

The description of hpdf\_threshold\_monitor\_high\_threshold is shown as below:

**Table 3-415. Function hpdf\_threshold\_monitor\_high\_threshold**

<b>Function name</b>	hpdf_threshold_monitor_high_threshold
<b>Function prototype</b>	void hpdf_threshold_monitor_channel(hpdf_filter_enum filtery, uint32_t channel);
<b>Function descriptions</b>	configure threshold monitor high threshold value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Input parameter{in}</b>	
<b>value</b>	high threshold value(-8388608~8388607)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure threshold monitor high threshold value */  
  
hpdf_threshold_monitor_high_threshold(FLT0, 32767);
```

## hpdf\_threshold\_monitor\_low\_threshold

The description of hpdf\_threshold\_monitor\_low\_threshold is shown as below:

**Table 3-416. Function hpdf\_threshold\_monitor\_low\_threshold**

<b>Function name</b>	hpdf_threshold_monitor_low_threshold
<b>Function prototype</b>	void hpdf_threshold_monitor_low_threshold(hpdf_filter_enum filtery, int32_t value);
<b>Function descriptions</b>	configure threshold monitor low threshold value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Input parameter{in}</b>	
<b>value</b>	low threshold value(-8388608~8388607)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure threshold monitor low threshold value */  
  
hpdf_threshold_monitor_low_threshold(FLT0, -32768);
```

## hpdf\_high\_threshold\_break\_signal

The description of hpdf\_high\_threshold\_break\_signal is shown as below:

**Table 3-417. Function hpdf\_high\_threshold\_break\_signal**

<b>Function name</b>	hpdf_high_threshold_break_signal
<b>Function prototype</b>	void hpdf_high_threshold_break_signal(hpdf_filter_enum filtery, uint32_t break_signal);
<b>Function descriptions</b>	configure threshold monitor high threshold event break signal
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>

Input parameter{in}	
<b>break_signal</b>	HPDF break signal
<i>NO_TM_HT_BREAK</i>	break signal is not distributed to an threshold monitor high threshold event
<i>TM_HT_BREAK0</i>	break signal 0 is distributed to an threshold monitor high threshold event
<i>TM_HT_BREAK1</i>	break signal 1 is distributed to an threshold monitor high threshold event
<i>TM_HT_BREAK0_1</i>	break signal 0 and 1 is distributed to an threshold monitor high threshold event
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure threshold monitor high threshold event break signal */
```

```
hpdf_high_threshold_break_signal(FTL0, TM_HT_BREAK0_1);
```

### hpdf\_low\_threshold\_break\_signal

The description of hpdf\_low\_threshold\_break\_signal is shown as below:

**Table 3-418. Function hpdf\_low\_threshold\_break\_signal**

<b>Function name</b>	hpdf_low_threshold_break_signal
<b>Function prototype</b>	void hpdf_low_threshold_break_signal(hpdf_filter_enum filtery, uint32_t break_signal);
<b>Function descriptions</b>	configure threshold monitor low threshold event break signal
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
Input parameter{in}	
<b>break_signal</b>	HPDF break signal
<i>NO_TM_LT_BREAK</i>	break signal is not distributed to an threshold monitor low threshold event
<i>TM_LT_BREAK0</i>	break signal 0 is distributed to an threshold monitor low threshold event
<i>TM_LT_BREAK1</i>	break signal 1 is distributed to an threshold monitor low threshold event
<i>TM_LT_BREAK0_1</i>	break signal 0 and 1 is distributed to an threshold monitor low threshold event
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure threshold monitor low threshold event break signal */
```

```
hpdf_low_threshold_break_signal(FTL0, TM_LT_BREAK0_1);
```

## hpdf\_extremes\_monitor\_channel

The description of hpdf\_extremes\_monitor\_channel is shown as below:

**Table 3-419. Function hpdf\_extremes\_monitor\_channel**

<b>Function name</b>	hpdf_extremes_monitor_channel
<b>Function prototype</b>	void hpdf_extremes_monitor_channel(hpdf_filter_enum filtery, uint32_t channel);
<b>Function descriptions</b>	configure extremes monitor channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Input parameter{in}</b>	
<b>channel</b>	which channel use extremes monitor y (y=0,1)
<i>EM_CHANNEL_DISABLE</i>	extremes monitor y does not accept data from channel 0 and channel 1
<i>EM_CHANNEL0</i>	extremes monitor y accepts data from channel 0
<i>EM_CHANNEL1</i>	extremes monitor y accepts data from channel 1
<i>EM_CHANNEL0_1</i>	extremes monitor y accepts data from channel 0 and channel 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure extremes monitor channel */
```

```
hpdf_extremes_monitor_channel(FTL0, EM_CHANNEL0_1);
```

## hpdf\_extremes\_monitor\_maximum\_get

The description of hpdf\_extremes\_monitor\_maximum\_get is shown as below:

**Table 3-420. Function hpdf\_extremes\_monitor\_maximum\_get**

<b>Function name</b>	hpdf_extremes_monitor_maximum_get
<b>Function prototype</b>	int32_t hpdf_extremes_monitor_maximum_get(hpdf_filter_enum filtery);
<b>Function descriptions</b>	get the extremes monitor maximum value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>int32_t</b>	the maximum value

Example:

```
int32_t vlaue;
```

```
vlaue = hpdf_extremes_monitor_maximum_get(FTL0);
```

### hpdf\_extremes\_monitor\_minimum\_get

The description of hpdf\_extremes\_monitor\_minimum\_get is shown as below:

**Table 3-421. Function hpdf\_extremes\_monitor\_minimum\_get**

<b>Function name</b>	hpdf_extremes_monitor_minimum_get
<b>Function prototype</b>	int32_t hpdf_extremes_monitor_minimum_get(hpdf_filter_enum filtery);
<b>Function descriptions</b>	get the extremes monitor minimum value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>int32_t</b>	the minimum value

Example:

```
/* get the extremes monitor minimum value */
```

```
int32_t vlaue;
```

```
vlaue = hpdf_extremes_monitor_minimum_get(FTL0,);
```

### hpdf\_rc\_continuous\_disable

The description of hpdf\_rc\_continuous\_disable is shown as below:

**Table 3-422. Function hpdf\_rc\_continuous\_disable**

<b>Function name</b>	hpdf_rc_continuous_disable
<b>Function prototype</b>	void hpdf_rc_continuous_disable(hpdf_filter_enum filtery);
<b>Function descriptions</b>	disable regular conversions continuous mode
<b>Precondition</b>	-

The called functions	-
Input parameter{in}	
filtery	The filter of HPDF module
FLTy(y=0..1)	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable regular conversions continuous mode */
```

```
hpdf_rc_continuous_disable(FTL0);
```

### hpdf\_rc\_continuous\_enable

The description of hpdf\_rc\_continuous\_enable is shown as below:

**Table 3-423. Function hpdf\_rc\_continuous\_enable**

Function name	hpdf_rc_continuous_enable
Function prototype	void hpdf_rc_continuous_enable(hpdf_filter_enum filtery);
Function descriptions	enable regular conversions continuous mode
Precondition	-
The called functions	-
Input parameter{in}	
filtery	The filter of HPDF module
FLTy(y=0..1)	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable regular conversions continuous mode */
```

```
hpdf_rc_continuous_enable(FTL0);
```

### hpdf\_rc\_start\_by\_software

The description of hpdf\_rc\_start\_by\_software is shown as below:

**Table 3-424. Function hpdf\_rc\_start\_by\_software**

Function name	hpdf_rc_start_by_software
Function prototype	void hpdf_rc_start_by_software(hpdf_filter_enum filtery);
Function descriptions	start regular channel conversion by software
Precondition	-



<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* start regular channel conversion by software */
```

```
hpdf_rc_start_by_software(FTL0);
```

### hpdf\_rc\_syn\_disable

The description of hpdf\_rc\_syn\_disable is shown as below:

**Table 3-425. Function hpdf\_rc\_syn\_disable**

<b>Function name</b>	hpdf_rc_syn_disable
<b>Function prototype</b>	void hpdf_rc_syn_disable(hpdf_filter_enum filtery);
<b>Function descriptions</b>	disable regular conversion synchronously
<b>Precondition</b>	disable FLTy(y=0..1)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable regular conversion synchronously */
```

```
hpdf_rc_syn_disable(FTL0);
```

### hpdf\_rc\_syn\_enable

The description of hpdf\_rc\_syn\_enable is shown as below:

**Table 3-426. Function hpdf\_rc\_syn\_enable**

<b>Function name</b>	hpdf_rc_syn_enable
<b>Function prototype</b>	void hpdf_rc_syn_enable(hpdf_filter_enum filtery);
<b>Function descriptions</b>	enable regular conversion synchronously
<b>Precondition</b>	disable FLTy(y=0..1)

The called functions	-
Input parameter{in}	
filtery	The filter of HPDF module
FLTy(y=0..1)	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable regular conversion synchronously */
```

```
hpdf_rc_syn_enable(FTL0);
```

### hpdf\_rc\_dma\_disable

The description of hpdf\_rc\_dma\_disable is shown as below:

**Table 3-427. Function hpdf\_rc\_dma\_disable**

Function name	hpdf_rc_dma_disable
Function prototype	void hpdf_rc_dma_disable(hpdf_filter_enum filtery);
Function descriptions	disable regular conversion DMA channel
Precondition	disable FLTy(y=0..1)
The called functions	-
Input parameter{in}	
filtery	The filter of HPDF module
FLTy(y=0..1)	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable regular conversion DMA channel */
```

```
hpdf_rc_dma_disable(FTL0);
```

### hpdf\_rc\_dma\_enable

The description of hpdf\_rc\_dma\_enable is shown as below:

**Table 3-428. Function hpdf\_rc\_dma\_enable**

Function name	hpdf_rc_dma_enable
Function prototype	void hpdf_rc_dma_enable(hpdf_filter_enum filtery);
Function descriptions	enable regular conversion DMA channel
Precondition	disable FLTy(y=0..1)

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable regular conversion DMA channel */
```

```
hpdf_rc_dma_enable(FTL0);
```

### hpdf\_rc\_channel\_config

The description of hpdf\_rc\_channel\_config is shown as below:

**Table 3-429. Function hpdf\_rc\_channel\_config**

<b>Function name</b>	hpdf_rc_channel_config
<b>Function prototype</b>	void hpdf_rc_channel_config(hpdf_filter_enum filtery, hpdf_channel_enum channelx);
<b>Function descriptions</b>	configure regular conversion channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Input parameter{in}</b>	
<b>channelx</b>	The channel of HPDF module
<i>CHANNELx(x=0..1)</i>	select HPDF channel, refer to <a href="#">Table 3-360. Enum hpdf_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure regular conversion channel */
```

```
hpdf_rc_channel_config(FTL0, CHANNEL1);
```

### hpdf\_rc\_fast\_mode\_disable

The description of hpdf\_rc\_fast\_mode\_disable is shown as below:

Table 3-430. Function `hpdf_rc_fast_mode_disable`

Function name	<code>hpdf_rc_fast_mode_disable</code>
Function prototype	<code>void hpdf_rc_fast_mode_disable(hpdf_filter_enum filtery);</code>
Function descriptions	disable regular conversion fast conversion mode
Precondition	disable <code>FLTy(y=0..1)</code>
The called functions	-
Input parameter{in}	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum <code>hpdf_filter_enum</code></a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable regular conversion fast conversion mode */
hpdf_rc_fast_mode_disable(FTL0);
```

### `hpdf_rc_fast_mode_enable`

The description of `hpdf_rc_fast_mode_enable` is shown as below:

Table 3-431. Function `hpdf_rc_fast_mode_enable`

Function name	<code>hpdf_rc_fast_mode_enable</code>
Function prototype	<code>void hpdf_rc_fast_mode_enable(hpdf_filter_enum filtery);</code>
Function descriptions	enable regular conversion fast conversion mode
Precondition	disable <code>FLTy(y=0..1)</code>
The called functions	-
Input parameter{in}	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum <code>hpdf_filter_enum</code></a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable regular conversion fast conversion mode */
hpdf_rc_fast_mode_enable(FTL0);
```

### `hpdf_rc_data_get`

The description of `hpdf_rc_data_get` is shown as below:

**Table 3-432. Function hpdf\_rc\_data\_get**

<b>Function name</b>	hpdf_rc_data_get
<b>Function prototype</b>	int32_t hpdf_rc_data_get(hpdf_filter_enum filtery);
<b>Function descriptions</b>	get the regular conversion data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>int32_t</b>	regular conversions data

Example:

```
/* get the regular conversion data */
int32_t vlaue;
vlaue = hpdf_rc_data_get(FTL0);
```

## hpdf\_rc\_channel\_get

The description of hpdf\_rc\_channel\_get is shown as below:

**Table 3-433. Function hpdf\_rc\_channel\_get**

<b>Function name</b>	hpdf_rc_channel_get
<b>Function prototype</b>	uint8_t hpdf_rc_channel_get(hpdf_filter_enum filtery);
<b>Function descriptions</b>	get the channel of regular channel most recently converted
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	the channel

Example:

```
/* get the channel of regular channel most recently converted */
uint8_t channel;
channel = hpdf_rc_channel_get(FTL0);
```

## hpdf\_ic\_start\_by\_software

The description of hpdf\_ic\_start\_by\_software is shown as below:

**Table 3-434. Function hpdf\_ic\_start\_by\_software**

<b>Function name</b>	hpdf_ic_start_by_software
<b>Function prototype</b>	void hpdf_ic_start_by_software(hpdf_filter_enum filtery);
<b>Function descriptions</b>	start inserted channel conversion by software
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* start inserted channel conversion by software */
```

```
hpdf_ic_start_by_software(FTL0);
```

## hpdf\_ic\_syn\_disable

The description of hpdf\_ic\_syn\_disable is shown as below:

**Table 3-435. Function hpdf\_ic\_syn\_disable**

<b>Function name</b>	hpdf_ic_syn_disable
<b>Function prototype</b>	void hpdf_ic_syn_disable(hpdf_filter_enum filtery);
<b>Function descriptions</b>	disable inserted conversion synchronously
<b>Precondition</b>	disable FLTy(y=0..1)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable inserted conversion synchronously */
```

```
hpdf_ic_syn_disable(FTL0);
```

## hpdf\_ic\_syn\_enable

The description of hpdf\_ic\_syn\_enable is shown as below:

**Table 3-436. Function hpdf\_ic\_syn\_enable**

<b>Function name</b>	hpdf_ic_syn_enable
<b>Function prototype</b>	void hpdf_ic_syn_enable(hpdf_filter_enum filtery);
<b>Function descriptions</b>	enable inserted conversion synchronously
<b>Precondition</b>	disable FLTy(y=0..1)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable inserted conversion synchronously */
hpdf_ic_syn_enable(FTL0);
```

## hpdf\_ic\_dma\_disable

The description of hpdf\_ic\_dma\_disable is shown as below:

**Table 3-437. Function hpdf\_ic\_dma\_disable**

<b>Function name</b>	hpdf_ic_dma_disable
<b>Function prototype</b>	void hpdf_ic_dma_disable(hpdf_filter_enum filtery);
<b>Function descriptions</b>	disable inserted conversion DMA channel
<b>Precondition</b>	disable FLTy(y=0..1)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable inserted conversion DMA channel */
hpdf_ic_dma_disable(FTL0);
```

## hpdf\_ic\_dma\_enable

The description of hpdf\_ic\_dma\_enable is shown as below:

**Table 3-438. Function hpdf\_ic\_dma\_enable**

<b>Function name</b>	hpdf_ic_dma_enable
<b>Function prototype</b>	void hpdf_ic_dma_enable(hpdf_filter_enum filtery);
<b>Function descriptions</b>	enable inserted conversion DMA channel
<b>Precondition</b>	disable FLTy(y=0..1)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable inserted conversion DMA channel */
```

```
hpdf_ic_dma_enable(FTL0);
```

## hpdf\_ic\_scan\_mode\_disable

The description of hpdf\_ic\_scan\_mode\_disable is shown as below:

**Table 3-439. Function hpdf\_ic\_scan\_mode\_disable**

<b>Function name</b>	hpdf_ic_scan_mode_disable
<b>Function prototype</b>	void hpdf_ic_scan_mode_disable(hpdf_filter_enum filtery);
<b>Function descriptions</b>	disable scan conversion mode
<b>Precondition</b>	disable FLTy(y=0..1)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable scan conversion mode */
```

```
hpdf_ic_scan_mode_disable(FTL0);
```



## hpdf\_ic\_scan\_mode\_enable

The description of hpdf\_ic\_scan\_mode\_enable is shown as below:

**Table 3-440. Function hpdf\_ic\_scan\_mode\_enable**

<b>Function name</b>	hpdf_ic_scan_mode_enable
<b>Function prototype</b>	void hpdf_ic_scan_mode_enable(hpdf_filter_enum filtery);
<b>Function descriptions</b>	enable scan conversion mode
<b>Precondition</b>	disable FLTy(y=0..1)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable scan conversion mode */
hpdf_ic_scan_mode_enable(FTL0);
```

## hpdf\_ic\_trigger\_signal\_disbale

The description of hpdf\_ic\_trigger\_signal\_disbale is shown as below:

**Table 3-441. Function hpdf\_ic\_trigger\_signal\_disbale**

<b>Function name</b>	hpdf_ic_trigger_signal_disbale
<b>Function prototype</b>	void hpdf_ic_trigger_signal_disbale(hpdf_filter_enum filtery);
<b>Function descriptions</b>	disable inserted conversions trigger siganl
<b>Precondition</b>	disable FLTy(y=0..1)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable inserted conversions trigger siganl */
hpdf_ic_trigger_signal_disbale(FTL0);
```

## hpdf\_ic\_trigger\_signal\_config

The description of hpdf\_ic\_trigger\_signal\_config is shown as below:

**Table 3-442. Function hpdf\_ic\_trigger\_signal\_config**

<b>Function name</b>	hpdf_ic_trigger_signal_config
<b>Function prototype</b>	void hpdf_ic_trigger_signal_config(hpdf_filter_enum filtery, uint32_t trigger, uint32_t trigger_edge);
<b>Function descriptions</b>	configure inserted conversions trigger signal and trigger edge
<b>Precondition</b>	disable FLTy(y=0..1)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Input parameter{in}</b>	
<b>trigger</b>	inserted conversions trigger signal
<i>HPDF_ITRG0</i>	TIMER1_TRGO is selected to start inserted conversion
<i>HPDF_ITRG1</i>	TIMER2_TRGO is selected to start inserted conversion
<i>HPDF_ITRG2</i>	TIMER3_TRGO is selected to start inserted conversion
<i>HPDF_ITRG3</i>	TIMER4_TRGO is selected to start inserted conversion
<i>HPDF_ITRG24</i>	EXTI11 is selected to start inserted conversion
<i>HPDF_ITRG25</i>	EXTI15 is selected to start inserted conversion
<i>HPDF_ITRG26</i>	TIMER5_TRGO is selected to start inserted conversion
<b>Input parameter{in}</b>	
<b>trigger_edge</b>	inserted conversions trigger edge
<i>TRG_DISABLE</i>	disable trigger signal
<i>RISING_EDGE_TRG</i>	rising edge on the trigger signal
<i>FALLING_EDGE_TRG</i>	falling edge on the trigger signal
<i>EDGE_TRG</i>	edge (rising edges and falling edges) on the trigger signal
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure inserted conversions trigger signal and trigger edge */
hpdf_ic_trigger_signal_config(FTL0, HPDF_ITRG3, FALLING_EDGE_TRG);
```

## hpdf\_ic\_channel\_config

The description of hpdf\_ic\_channel\_config is shown as below:

**Table 3-443. Function hpdf\_ic\_channel\_config**

<b>Function name</b>	hpdf_ic_channel_config
----------------------	------------------------

<b>Function prototype</b>	void hpdf_ic_channel_config(hpdf_filter_enum filtery, uint32_t channel);
<b>Function descriptions</b>	configure inserted group conversions channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Input parameter{in}</b>	
<b>channel</b>	the HPDF channel belongs to inserted group
<i>IGCSEL_CHANNEL0</i>	channel0 belongs to the inserted group
<i>IGCSEL_CHANNEL1</i>	channel1 belongs to the inserted group
<i>IGCSEL_CHANNEL0_1</i>	channel0 and channel1 belongs to the inserted group
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure inserted group conversions channel */
hpdf_ic_channel_config(FTL0, IGCSEL_CHANNEL0_1);
```

## hpdf\_ic\_data\_get

The description of hpdf\_ic\_data\_get is shown as below:

**Table 3-444. Function hpdf\_ic\_data\_get**

<b>Function name</b>	hpdf_ic_data_get
<b>Function prototype</b>	int32_t hpdf_ic_data_get(hpdf_filter_enum filtery);
<b>Function descriptions</b>	get the inserted conversions data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>int32_t</b>	inserted conversions data

Example:

```
/* get the inserted conversions data */
int32_t vlaue;
```

```
vlaue = hpdf_ic_data_get(FTL0);
```

## hpdf\_ic\_channel\_get

The description of hpdf\_ic\_channel\_get is shown as below:

**Table 3-445. Function hpdf\_ic\_channel\_get**

<b>Function name</b>	hpdf_ic_channel_get
<b>Function prototype</b>	uint8_t hpdf_ic_channel_get(hpdf_filter_enum filtery);
<b>Function descriptions</b>	get the channel of inserted group channel most recently converted
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	the channel

Example:

```
/* get the channel of inserted group channel most recently converted */
```

```
uint8_t channel;
```

```
channel = hpdf_ic_channel_get(FTL0);
```

## hpdf\_flag\_get

The description of hpdf\_flag\_get is shown as below:

**Table 3-446. Function hpdf\_flag\_get**

<b>Function name</b>	hpdf_flag_get
<b>Function prototype</b>	FlagStatus hpdf_flag_get(hpdf_filter_enum filtery, hpdf_flag_enum flag);
<b>Function descriptions</b>	get the HPDF flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Input parameter{in}</b>	
<b>flag</b>	HPDF flags, refer to <a href="#">Table 3-362. Enum hpdf_flag_enum</a>
<i>HPDF_FLAG_FLTy_IC EF</i>	FLTy inserted conversion end flag
<i>HPDF_FLAG_FLTy_RC EF</i>	FLTy regular conversion end flag

<i>HPDF_FLAG_FLTy_IC DOF</i>	FLTy inserted conversion data overflow flag
<i>HPDF_FLAG_FLTy_RC DOF</i>	FLTy threshold monitor event occurred flag
<i>HPDF_FLAG_FLTy_TM EOF</i>	FLTy inserted conversion in progress flag
<i>HPDF_FLAG_FLTy_IC PF</i>	FLTy regular conversion in progress flag
<i>HPDF_FLAG_FLTy_RC PF</i>	clock signal is lost on channel 0 flag
<i>HPDF_FLAG_FLT0_CK LF0</i>	clock signal is lost on channel 1 flag
<i>HPDF_FLAG_FLT0_CK LF1</i>	malfunction event occurred on channel 0 flag
<i>HPDF_FLAG_FLT0_M MF0</i>	malfunction occurred on channel 1 flag
<i>HPDF_FLAG_FLT0_M MF1</i>	FLTy inserted channel most recently converted
<i>HPDF_FLAG_FLTy_RC HPDT</i>	FLTy inserted channel most recently converted
<i>HPDF_FLAG_FLTy_LT F0</i>	threshold monitor low threshold flag on channel 0 flag
<i>HPDF_FLAG_FLTy_LT F1</i>	threshold monitor low threshold flag on channel 1 flag
<i>HPDF_FLAG_FLTy_HT F0</i>	threshold monitor high threshold flag on channel 0 flag
<i>HPDF_FLAG_FLTy_HT F1</i>	threshold monitor high threshold flag on channel 1 flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the FLT0 threshold monitor event occurred flag */
```

```
hpdf_flag_get(FTL0, HPDF_FLAG_FLTy_TMEOF);
```

### hpdf\_flag\_clear

The description of hpdf\_flag\_clear is shown as below:

**Table 3-447. Function hpdf\_flag\_clear**

<b>Function name</b>	hpdf_flag_clear
----------------------	-----------------

<b>Function prototype</b>	void hpdf_flag_clear(hpdf_filter_enum filtery, hpdf_flag_enum flag);
<b>Function descriptions</b>	clear the HPDF flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Input parameter{in}</b>	
<b>flag</b>	HPDF flags, refer to <a href="#">Table 3-362. Enum hpdf_flag_enum</a>
<i>HPDF_FLAG_FLTy_IC EF</i>	FLTy inserted conversion end flag
<i>HPDF_FLAG_FLTy_RC EF</i>	FLTy regular conversion end flag
<i>HPDF_FLAG_FLTy_IC DOF</i>	FLTy inserted conversion data overflow flag
<i>HPDF_FLAG_FLTy_RC DOF</i>	FLTy threshold monitor event occurred flag
<i>HPDF_FLAG_FLTy_TM EOF</i>	FLTy inserted conversion in progress flag
<i>HPDF_FLAG_FLTy_IC PF</i>	FLTy regular conversion in progress flag
<i>HPDF_FLAG_FLTy_RC PF</i>	clock signal is lost on channel 0 flag
<i>HPDF_FLAG_FLT0_CK LF0</i>	clock signal is lost on channel 1 flag
<i>HPDF_FLAG_FLT0_CK LF1</i>	malfunction event occurred on channel 0 flag
<i>HPDF_FLAG_FLT0_M MF0</i>	malfunction occurred on channel 1 flag
<i>HPDF_FLAG_FLT0_M MF1</i>	FLTy inserted channel most recently converted
<i>HPDF_FLAG_FLTy_RC HPDT</i>	FLTy inserted channel most recently converted
<i>HPDF_FLAG_FLTy_LT F0</i>	threshold monitor low threshold flag on channel 0 flag
<i>HPDF_FLAG_FLTy_LT F1</i>	threshold monitor low threshold flag on channel 1 flag
<i>HPDF_FLAG_FLTy_HT F0</i>	threshold monitor high threshold flag on channel 0 flag
<i>HPDF_FLAG_FLTy_HT F1</i>	threshold monitor high threshold flag on channel 1 flag
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* clear the FLT0 threshold monitor event occurred flag */
```

```
hpdf_flag_clear(FTL0, HPDF_FLAG_FLTy_TMEOF);
```

## hpdf\_interrupt\_enable

The description of hpdf\_interrupt\_enable is shown as below:

**Table 3-448. Function hpdf\_interrupt\_enable**

Function name	hpdf_interrupt_enable
Function prototype	void hpdf_interrupt_enable(hpdf_filter_enum filtery, hpdf_interrput_enum interrupt);
Function descriptions	enable HPDF interrupt
Precondition	-
The called functions	-
Input parameter{in}	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
Input parameter{in}	
<b>interrupt</b>	HPDF interrupts, refer to <a href="#">Table 3-364. Enum hpdf_interrput_enum</a>
HPDF_INT_FLTy_ICEI E	FLTy inserted conversion end interrupt enable
HPDF_INT_FLTy_RCEI E	FLTy regular conversion end interrupt enable
HPDF_INT_FLTy_ICD OIE	FLTy inserted conversion data overflow interrupt enable
HPDF_INT_FLTy_RCD OIE	FLTy regular conversion data overflow interrupt enable
HPDF_INT_FLTy_TMIE	FLTy threshold monitor interrupt enable
HPDF_INT_FLT0_MMI E	malfunction monitor interrupt enable
HPDF_INT_FLT0_CKLI E	clock loss interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable FLT0 threshold monitor interrupt */
```

hpdf\_interrupt\_enable(FTL0, HPDF\_INT\_FLTy\_TME0IE);

## hpdf\_interrupt\_disable

The description of hpdf\_interrupt\_disable is shown as below:

**Table 3-449. Function hpdf\_interrupt\_disable**

<b>Function name</b>	hpdf_interrupt_disable
<b>Function prototype</b>	void hpdf_interrupt_disable(hpdf_filter_enum filtery, hpdf_interrput_enum interrupt);
<b>Function descriptions</b>	disable HPDF interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<i>FLTy(y=0..1)</i>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Input parameter{in}</b>	
<b>interrupt</b>	HPDF interrupts, refer to <a href="#">Table 3-364. Enum hpdf_interrput_enum</a>
HPDF_INT_FLTy_ICEIE	FLTy inserted conversion end interrupt enable
HPDF_INT_FLTy_RCEIE	FLTy regular conversion end interrupt enable
HPDF_INT_FLTy_ICDOIE	FLTy inserted conversion data overflow interrupt enable
HPDF_INT_FLTy_RCDIE	FLTy regular conversion data overflow interrupt enable
HPDF_INT_FLTy_TMIE	FLTy threshold monitor interrupt enable
HPDF_INT_FLT0_MMIE	malfunction monitor interrupt enable
HPDF_INT_FLT0_CKLIIE	clock loss interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable FLT0 threshold monitor interrupt */
```

```
hpdf_interrupt_disable(FTL0, HPDF_INT_FLT0_CKLIIE);
```

## hpdf\_interrupt\_flag\_get

The description of hpdf\_interrupt\_flag\_get is shown as below:



**Table 3-450. Function `hpdf_interrupt_flag_get`**

<b>Function name</b>	<code>hpdf_interrupt_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus hpdf_interrupt_flag_get(hpdf_filter_enum filtery, hpdf_interrupt_flag_enum int_flag);</code>
<b>Function descriptions</b>	get the HPDF interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<code>FLTy(y=0..1)</code>	select HPDF filter, refer to <a href="#">Table 3-361. Enum <code>hpdf_filter_enum</code></a>
<b>Input parameter{in}</b>	
<b>int_flag</b>	HPDF flags, refer to <a href="#">Table 3-363. Enum <code>hpdf_interrput_flag_enum</code></a>
<code>HPDF_INT_FLAG_FLTy_ICEF</code>	FLTy inserted conversion end interrupt flag
<code>HPDF_INT_FLAG_FLTy_RCEF</code>	FLTy regular conversion end interrupt flag
<code>HPDF_INT_FLAG_FLTy_ICDOF</code>	FLTy inserted conversion data overflow interrupt flag
<code>HPDF_INT_FLAG_FLTy_RCDOF</code>	FLTy regular conversion data overflow interrupt flag
<code>HPDF_INT_FLAG_FLTy_TMEOF</code>	FLTy threshold monitor event occurred interrupt flag
<code>HPDF_INT_FLAG_FLT0_CKLF0</code>	clock signal is lost on channel 0 interrupt flag
<code>HPDF_INT_FLAG_FLT0_CKLF1</code>	clock signal is lost on channel 1 interrupt flag
<code>HPDF_INT_FLAG_FLT0_MMF0</code>	malfunction event occurred on channel 0 interrupt flag
<code>HPDF_INT_FLAG_FLT0_MMF1</code>	malfunction event occurred on channel 1 interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the the clock loss interrupt flag */
```

```
hpdf_interrupt_flag_get (FTL0, HPDF_INT_FLAG_FLT0_CKLF0);
```

## **`hpdf_interrupt_flag_clear`**

The description of `hpdf_interrupt_flag_clear` is shown as below:

Table 3-451. Function `hpdf_interrupt_flag_clear`

<b>Function name</b>	<code>hpdf_interrupt_flag_clear</code>
<b>Function prototype</b>	<code>void hpdf_interrupt_flag_clear(hpdf_filter_enum filtery, hpdf_interrupt_flag_enum int_flag);</code>
<b>Function descriptions</b>	clear the HPDF interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>filtery</b>	The filter of HPDF module
<code>FLTy(y=0..1)</code>	select HPDF filter, refer to <a href="#">Table 3-361. Enum hpdf_filter_enum</a>
<b>Input parameter{in}</b>	
<b>int_flag</b>	HPDF flags, refer to <a href="#">Table 3-363. Enum hpdf_interrupt_flag_enum</a>
<code>HPDF_INT_FLAG_FLTy_ICEF</code>	FLTy inserted conversion end interrupt flag
<code>HPDF_INT_FLAG_FLTy_RCEF</code>	FLTy regular conversion end interrupt flag
<code>HPDF_INT_FLAG_FLTy_ICDOF</code>	FLTy inserted conversion data overflow interrupt flag
<code>HPDF_INT_FLAG_FLTy_RCDOF</code>	FLTy regular conversion data overflow interrupt flag
<code>HPDF_INT_FLAG_FLTy_TMEOF</code>	FLTy threshold monitor event occurred interrupt flag
<code>HPDF_INT_FLAG_FLT0_CKLF0</code>	clock signal is lost on channel 0 interrupt flag
<code>HPDF_INT_FLAG_FLT0_CKLF1</code>	clock signal is lost on channel 1 interrupt flag
<code>HPDF_INT_FLAG_FLT0_MMF0</code>	malfunction event occurred on channel 0 interrupt flag
<code>HPDF_INT_FLAG_FLT0_MMF1</code>	malfunction event occurred on channel 1 interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the the clock loss interrupt flag */
```

```
hpdf_interrupt_flag_get(FTL0, HPDF_INT_FLAG_FLT0_CKLF0);
```

## 3.16. I2C

The I2C (inter-integrated circuit) module provides an I2C interface which is an industry

standard two-line serial interface for MCU to communicate with external I2C interface. The I2C registers are listed in chapter [3.16.1](#), the I2C firmware functions are introduced in chapter [3.16.2](#).

### 3.16.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

**Table 3-452. 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_TIMING	Timing register
I2C_TIMEOUT	Timeout register
I2C_STAT	Status register
I2C_STATC	I2C status clear register
I2C_PEC	PEC register
I2C_RDATA	Receive data register
I2C_TDATA	Transmit data register
I2C_CTL2	Control register 2

### 3.16.2. Descriptions of Peripheral functions

I2C firmware functions are listed in the table shown as below:

**Table 3-453. I2C firmware function**

Function name	Function description
i2c_deinit	reset I2C
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_master_clock_config	configure the SCL high and low period of clock in master mode
i2c_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

Function name	Function description
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_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_address_config	configure i2c slave address
i2c_address_bit_compare_config	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
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_received_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_wakeup_from_deepsleep_enable	enable wakeup from Deep-sleep mode
i2c_wakeup_from_deepsleep_disable	disable wakeup from Deep-sleep mode
i2c_enable	enable I2C
i2c_disable	disable I2C
i2c_start_on_bus	generate a START condition on I2C bus
i2c_stop_on_bus	generate a STOP condition on I2C bus
i2c_data_transmit	I2C transmit data
i2c_data_receive	I2C receive data
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
i2c_dma_enable	enable I2C DMA for transmission or reception
i2c_dma_disable	disable I2C DMA for transmission or reception
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_smbus_alert_enable	enable SMBus Alert
i2c_smbus_alert_disable	disable SMBus Alert
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

Function name	Function description
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
i2c_flag_get	get I2C flag status
i2c_flag_clear	clear I2C flag status
i2c_interrupt_enable	enable I2C interrupt
i2c_interrupt_disable	disable I2C interrupt
i2c_interrupt_flag_get	get I2C interrupt flag status
i2c_interrupt_flag_clear	clear I2C interrupt flag status

## Enum i2c\_interrupt\_flag\_enum

**Table 3-454. i2c\_interrupt\_flag\_enum**

Member name	Function description
I2C_INT_FLAG_TI	transmit interrupt flag
I2C_INT_FLAG_RBNE	I2C_RDATA is not empty during receiving interrupt flag
I2C_INT_FLAG_ADDSEND	address received matches in slave mode interrupt flag
I2C_INT_FLAG_NACK	not acknowledge interrupt flag
I2C_INT_FLAG_STPDET	stop condition detected in slave mode interrupt flag
I2C_INT_FLAG_TC	transfer complete in master mode interrupt flag
I2C_INT_FLAG_TCR	transfer complete reload interrupt flag
I2C_INT_FLAG_BERR	bus error interrupt flag
I2C_INT_FLAG_LOSTARB	arbitration lost interrupt flag
I2C_INT_FLAG_OUERR	overflow/underrun error in slave mode interrupt flag
I2C_INT_FLAG_PECERR	PEC error interrupt flag
I2C_INT_FLAG_TIMEOUT	timeout interrupt flag
I2C_INT_FLAG_SMBALT	SMBus Alert interrupt flag

## i2c\_deinit

The description of i2c\_deinit is shown as below:

**Table 3-455. Function i2c\_deinit**

Function name	i2c_deinit
Function prototype	void i2c_deinit(uint32_t i2c_periph);
Function descriptions	reset I2C
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable

Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset I2C0 */
```

```
i2c_deinit(I2C0);
```

## i2c\_timing\_config

The description of i2c\_timing\_config is shown as below:

**Table 3-456. Function i2c\_timing\_config**

<b>Function name</b>	i2c_timing_config
<b>Function prototype</b>	void i2c_timing_config(uint32_t i2c_periph, uint32_t psc, uint32_t scl_dely, uint32_t sda_dely);
<b>Function descriptions</b>	configure the timing parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
<b>psc</b>	0-0xf, timing prescaler
Input parameter{in}	
<b>scl_dely</b>	0-0xf, data setup time
Input parameter{in}	
<b>sda_dely</b>	0-0xf, data hold time
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the timing parameters */
```

```
i2c_timing_config(I2C0, 0x1, 0x2, 0x1);
```

## i2c\_digital\_noise\_filter\_config

The description of i2c\_digital\_noise\_filter\_config is shown as below:

**Table 3-457. Function i2c\_digital\_noise\_filter\_config**

<b>Function name</b>	i2c_digital_noise_filter_config
<b>Function prototype</b>	void i2c_digital_noise_filter_config(uint32_t i2c_periph, uint32_t filter_length);
<b>Function descriptions</b>	configure digital noise filter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>filter_length</b>	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 tI2CCLK
<i>FILTER_LENGTH_2</i>	digital filter is enabled and filter spikes with a length of up to 2 tI2CCLK
<i>FILTER_LENGTH_3</i>	digital filter is enabled and filter spikes with a length of up to 3 tI2CCLK
<i>FILTER_LENGTH_4</i>	digital filter is enabled and filter spikes with a length of up to 4 tI2CCLK
<i>FILTER_LENGTH_5</i>	digital filter is enabled and filter spikes with a length of up to 5 tI2CCLK
<i>FILTER_LENGTH_6</i>	digital filter is enabled and filter spikes with a length of up to 6 tI2CCLK
<i>FILTER_LENGTH_7</i>	digital filter is enabled and filter spikes with a length of up to 7 tI2CCLK
<i>FILTER_LENGTH_8</i>	digital filter is enabled and filter spikes with a length of up to 8 tI2CCLK
<i>FILTER_LENGTH_9</i>	digital filter is enabled and filter spikes with a length of up to 9 tI2CCLK
<i>FILTER_LENGTH_10</i>	digital filter is enabled and filter spikes with a length of up to 10 tI2CCLK
<i>FILTER_LENGTH_11</i>	digital filter is enabled and filter spikes with a length of up to 11 tI2CCLK
<i>FILTER_LENGTH_12</i>	digital filter is enabled and filter spikes with a length of up to 12 tI2CCLK
<i>FILTER_LENGTH_13</i>	digital filter is enabled and filter spikes with a length of up to 13 tI2CCLK
<i>FILTER_LENGTH_14</i>	digital filter is enabled and filter spikes with a length of up to 14 tI2CCLK
<i>FILTER_LENGTH_15</i>	digital filter is enabled and filter spikes with a length of up to 15 tI2CCLK
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 digital filter filters spikes with a length of up to 1 tI2CCLK */
```

```
i2c_digital_noise_filter_config(I2C0, FILTER_LENGTH_1);
```

## i2c\_analog\_noise\_filter\_enable

The description of i2c\_analog\_noise\_filter\_enable is shown as below:

**Table 3-458. Function i2c\_analog\_noise\_filter\_enable**

<b>Function name</b>	i2c_analog_noise_filter_enable
<b>Function prototype</b>	void i2c_analog_noise_filter_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable analog noise filter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable analog noise filter */
i2c_analog_noise_filter_enable(I2C0);
```

## i2c\_analog\_noise\_filter\_disable

The description of i2c\_analog\_noise\_filter\_disable is shown as below:

**Table 3-459. Function i2c\_analog\_noise\_filter\_disable**

<b>Function name</b>	i2c_analog_noise_filter_disable
<b>Function prototype</b>	void i2c_analog_noise_filter_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable analog noise filter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable analog noise filter */
i2c_analog_noise_filter_disable(I2C0);
```



## i2c\_master\_clock\_config

The description of i2c\_master\_clock\_config is shown as below:

**Table 3-460. Function i2c\_master\_clock\_config**

<b>Function name</b>	i2c_master_clock_config
<b>Function prototype</b>	void i2c_master_clock_config(uint32_t i2c_periph, uint32_t sclh, uint32_t scll);
<b>Function descriptions</b>	configure the SCL high and low period of clock in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>sclh</b>	0-0xff, SCL high period
<b>Input parameter{in}</b>	
<b>scll</b>	0-0xff, SCL low period
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the SCL and SDA period of clock in master mode */
```

```
i2c_master_clock_config (I2C0, 0x0f, 0x0f);
```

## i2c\_master\_addressing

The description of i2c\_master\_addressing is shown as below:

**Table 3-461. Function i2c\_master\_addressing**

<b>Function name</b>	i2c_master_addressing
<b>Function prototype</b>	void i2c_master_addressing(uint32_t i2c_periph, uint32_t address, uint32_t trans_direction);
<b>Function descriptions</b>	configure i2c slave addresss and transfer direction in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>address</b>	0-0x3FF except reserved address, I2C slave address to be sent
<b>Input parameter{in}</b>	

<b>trans_direction</b>	I2C transfer direction in master mode
<i>I2C_MASTER_TRANSMIT</i>	master transmit
<i>I2C_MASTER_RECEIVE</i>	master receive
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* send slave address to I2C bus */
```

```
i2c_master_addressing (I2C0, 0x82, I2C_MASTER_TRANSMIT);
```

### i2c\_address10\_header\_enable

The description of i2c\_address10\_header\_enable is shown as below:

**Table 3-462. Function i2c\_address10\_header\_enable**

<b>Function name</b>	i2c_address10_header_enable
<b>Function prototype</b>	void i2c_address10_header_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	10-bit address header executes read direction only in master receive mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 10-bit address header executes read direction only in master receive mode */
```

```
i2c_address10_header_enable(I2C0);
```

### i2c\_address10\_header\_disable

The description of i2c\_address10\_header\_disable is shown as below:

**Table 3-463. Function i2c\_address10\_header\_disable**

<b>Function name</b>	i2c_address10_header_disable
<b>Function prototype</b>	void i2c_address10_header_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	10-bit address header executes complete sequence in master receive mode

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 10-bit address header executes complete sequence in master receive mode */
```

```
i2c_address10_header_disable(I2C0);
```

### i2c\_address10\_enable

The description of i2c\_address10\_enable is shown as below:

**Table 3-464. Function i2c\_address10\_enable**

<b>Function name</b>	i2c_address10_enable
<b>Function prototype</b>	void i2c_address10_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable 10-bit addressing mode in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable 10-bit addressing mode in master mode */
```

```
i2c_address10_enable(I2C0);
```

### i2c\_address10\_disable

The description of i2c\_address10\_disable is shown as below:

**Table 3-465. Function i2c\_address10\_disable**

<b>Function name</b>	i2c_address10_disable
<b>Function prototype</b>	void i2c_address10_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable 10-bit addressing mode in master mode

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable 10-bit addressing mode in master mode */
```

```
i2c_address10_disable(I2C0);
```

### i2c\_automatic\_end\_enable

The description of i2c\_automatic\_end\_enable is shown as below:

**Table 3-466. Function i2c\_automatic\_end\_enable**

<b>Function name</b>	i2c_automatic_end_enable
<b>Function prototype</b>	void i2c_automatic_end_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable I2C automatic end mode in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_enable(I2C0);
```

### i2c\_automatic\_end\_disable

The description of i2c\_automatic\_end\_disable is shown as below:

**Table 3-467. Function i2c\_automatic\_end\_disable**

<b>Function name</b>	i2c_automatic_end_disable
<b>Function prototype</b>	void i2c_automatic_end_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable I2C automatic end mode in master mode

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_disable(I2C0);
```

### i2c\_slave\_response\_to\_gcall\_enable

The description of i2c\_slave\_response\_to\_gcall\_enable is shown as below:

**Table 3-468. Function i2c\_slave\_response\_to\_gcall\_enable**

<b>Function name</b>	i2c_slave_response_to_gcall_enable
<b>Function prototype</b>	void i2c_slave_response_to_gcall_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable the response to a general call
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-469. Function i2c\_slave\_response\_to\_gcall\_disable**

<b>Function name</b>	i2c_slave_response_to_gcall_disable
<b>Function prototype</b>	void i2c_slave_response_to_gcall_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable the response to a general call

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-470. Function i2c\_stretch\_scl\_low\_enable**

<b>Function name</b>	i2c_stretch_scl_low_enable
<b>Function prototype</b>	void i2c_stretch_scl_low_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable to stretch SCL low when data is not ready in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-471. Function i2c\_stretch\_scl\_low\_disable**

<b>Function name</b>	i2c_stretch_scl_low_disable
<b>Function prototype</b>	void i2c_stretch_scl_low_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable to stretch SCL low when data is not ready in slave mode

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_disable(I2C0);
```

## i2c\_address\_config

The description of i2c\_address\_config is shown as below:

**Table 3-472. Function i2c\_address\_config**

<b>Function name</b>	i2c_address_config
<b>Function prototype</b>	void i2c_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_format);
<b>Function descriptions</b>	configure i2c slave address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>address</b>	I2C address
<b>Input parameter{in}</b>	
<b>addr_format</b>	7bits or 10bits
<i>I2C_ADDFORMAT_7BITS</i>	7bits
<i>I2C_ADDFORMAT_10BITS</i>	10bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure i2c slave address */
```

i2c\_address\_config (I2C0, 0x82, I2C\_ADDFORMAT\_7BITS);

## i2c\_address\_bit\_compare\_config

The description of i2c\_address\_bit\_compare\_config is shown as below:

**Table 3-473. Function i2c\_address\_bit\_compare\_config**

<b>Function name</b>	i2c_address_bit_compare_config
<b>Function prototype</b>	void i2c_address_bit_compare_config(uint32_t i2c_periph, uint32_t compare_bits);
<b>Function descriptions</b>	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>compare_bits</b>	the bits need to compare
<i>ADDRESS_BIT1_COMPARE</i>	address bit1 needs compare
<i>ADDRESS_BIT2_COMPARE</i>	address bit2 needs compare
<i>ADDRESS_BIT3_COMPARE</i>	address bit3 needs compare
<i>ADDRESS_BIT4_COMPARE</i>	address bit4 needs compare
<i>ADDRESS_BIT5_COMPARE</i>	address bit5 needs compare
<i>ADDRESS_BIT6_COMPARE</i>	address bit6 needs compare
<i>ADDRESS_BIT7_COMPARE</i>	address bit7 needs compare
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

/\* bit 1 of ADDRESS[7:1] need to compare with the incoming address byte \*/

i2c\_address\_bit\_compare\_config(I2C0, ADDRESS\_BIT1\_COMPARE);



## i2c\_address\_disable

The description of i2c\_address\_disable is shown as below:

**Table 3-474. Function i2c\_address\_disable**

<b>Function name</b>	i2c_address_disable
<b>Function prototype</b>	void i2c_address_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable i2c address in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable i2c address in slave mode */
```

```
i2c_address_disable(I2C0);
```

## i2c\_second\_address\_config

The description of i2c\_second\_address\_config is shown as below:

**Table 3-475. Function i2c\_second\_address\_config**

<b>Function name</b>	i2c_second_address_config
<b>Function prototype</b>	void i2c_second_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_mask);
<b>Function descriptions</b>	configure i2c second slave address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>address</b>	I2C address
<b>Input parameter{in}</b>	
<b>addr_mask</b>	the bits not need to compare
<i>ADDRESS2_NO_MASK</i>	no mask, all the bits must be compared
<i>ADDRESS2_MASK_BIT1</i>	ADDRESS2[1] is masked, only ADDRESS2[7:2] are compared

ADDRESS2_MASK_BIT1_2	ADDRESS2[2:1] is masked, only ADDRESS2[7:3] are compared
ADDRESS2_MASK_BIT1_3	ADDRESS2[3:1] is masked, only ADDRESS2[7:4] are compared
ADDRESS2_MASK_BIT1_4	ADDRESS2[4:1] is masked, only ADDRESS2[7:5] are compared
ADDRESS2_MASK_BIT1_5	ADDRESS2[5:1] is masked, only ADDRESS2[7:6] are compared
ADDRESS2_MASK_BIT1_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 (I2C0, 0x82, ADDRESS2_MASK_BIT1_2);
```

### i2c\_second\_address\_disable

The description of i2c\_second\_address\_disable is shown as below:

**Table 3-476. 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=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable i2c second address in slave mode */
```

```
i2c_second_address_disable(I2C0);
```

## i2c\_receved\_address\_get

The description of i2c\_receved\_address\_get is shown as below:

**Table 3-477. Function i2c\_receved\_address\_get**

<b>Function name</b>	i2c_receved_address_get
<b>Function prototype</b>	uint32_t i2c_receved_address_get(uint32_t i2c_periph);
<b>Function descriptions</b>	get received match address in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	0x00..0x7F

Example:

```
/* get received match address in slave mode */
```

```
uint32_t address;
```

```
address = i2c_receved_address_get(I2C0);
```

## i2c\_slave\_byte\_control\_enable

The description of i2c\_slave\_byte\_control\_enable is shown as below:

**Table 3-478. Function i2c\_slave\_byte\_control\_enable**

<b>Function name</b>	i2c_slave_byte_control_enable
<b>Function prototype</b>	void i2c_slave_byte_control_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable slave byte control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable slave byte control */
```

i2c\_slave\_byte\_control\_enable(I2C0);

## i2c\_slave\_byte\_control\_disable

The description of i2c\_slave\_byte\_control\_disable is shown as below:

**Table 3-479. Function i2c\_slave\_byte\_control\_disable**

<b>Function name</b>	i2c_slave_byte_control_disable
<b>Function prototype</b>	void i2c_slave_byte_control_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable slave byte control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable slave byte control */
```

```
i2c_slave_byte_control_disable(I2C0);
```

## i2c\_nack\_enable

The description of i2c\_nack\_enable is shown as below:

**Table 3-480. Function i2c\_nack\_enable**

<b>Function name</b>	i2c_nack_enable
<b>Function prototype</b>	void i2c_nack_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a NACK in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* generate a NACK in slave mode */
```

i2c\_nack\_enable(I2C0);

## i2c\_nack\_disable

The description of i2c\_nack\_disable is shown as below:

**Table 3-481. Function i2c\_nack\_disable**

<b>Function name</b>	i2c_nack_disable
<b>Function prototype</b>	void i2c_nack_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a ACK in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* generate a ACK in slave mode */
```

```
i2c_nack_disable(I2C0);
```

## i2c\_wakeup\_from\_deepsleep\_enable

The description of i2c\_wakeup\_from\_deepsleep\_enable is shown as below:

**Table 3-482. Function i2c\_wakeup\_from\_deepsleep\_enable**

<b>Function name</b>	i2c_wakeup_from_deepsleep_enable
<b>Function prototype</b>	void i2c_wakeup_from_deepsleep_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable wakeup from Deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable wakeup from Deep-sleep mode */
```

i2c\_wakeup\_from\_deepsleep\_enable(I2C0);

## i2c\_wakeup\_from\_deepsleep\_disable

The description of i2c\_wakeup\_from\_deepsleep\_disable is shown as below:

**Table 3-483. Function i2c\_wakeup\_from\_deepsleep\_disable**

<b>Function name</b>	i2c_wakeup_from_deepsleep_disable
<b>Function prototype</b>	void i2c_wakeup_from_deepsleep_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable wakeup from Deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable wakeup from Deep-sleep mode */
```

```
i2c_wakeup_from_deepsleep_disable(I2C0);
```

## i2c\_enable

The description of i2c\_enable is shown as below:

**Table 3-484. Function i2c\_enable**

<b>Function name</b>	i2c_enable
<b>Function prototype</b>	void i2c_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 */
```

i2c\_enable(I2C0);

## i2c\_disable

The description of i2c\_disable is shown as below:

**Table 3-485. Function i2c\_disable**

<b>Function name</b>	i2c_disable
<b>Function prototype</b>	void i2c_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 */
```

```
i2c_disable(I2C0);
```

## i2c\_start\_on\_bus

The description of i2c\_start\_on\_bus is shown as below:

**Table 3-486. Function i2c\_start\_on\_bus**

<b>Function name</b>	i2c_start_on_bus
<b>Function prototype</b>	void i2c_start_on_bus(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a START condition on I2C bus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-487. Function i2c\_stop\_on\_bus**

<b>Function name</b>	i2c_stop_on_bus
<b>Function prototype</b>	void i2c_stop_on_bus(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a STOP condition on I2C bus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 generate a STOP condition to I2C bus */
```

```
i2c_stop_on_bus(I2C0);
```

## i2c\_data\_transmit

The description of i2c\_data\_transmit is shown as below:

**Table 3-488. Function i2c\_data\_transmit**

<b>Function name</b>	i2c_data_transmit
<b>Function prototype</b>	void i2c_data_transmit(uint32_t i2c_periph, uint32_t data);
<b>Function descriptions</b>	I2C transmit data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>data</b>	transmit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:



```
/* I2C0 transmit data */
```

```
i2c_data_transmit (I2C0, 0x80);
```

### i2c\_data\_receive

The description of i2c\_data\_receive is shown as below:

**Table 3-489. Function i2c\_data\_receive**

<b>Function name</b>	i2c_data_receive
<b>Function prototype</b>	uint32_t i2c_data_receive(uint32_t i2c_periph);
<b>Function descriptions</b>	I2C receive data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	0x0000..0x00FF

Example:

```
/* I2C0 receive data */
```

```
uint32_t i2c_receiver;
```

```
i2c_receiver = i2c_data_receive(I2C0);
```

### i2c\_reload\_enable

The description of i2c\_reload\_enable is shown as below:

**Table 3-490. Function i2c\_reload\_enable**

<b>Function name</b>	i2c_reload_enable
<b>Function prototype</b>	void i2c_reload_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable I2C reload mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C reload mode */
```

```
i2c_reload_enable(I2C0);
```

### i2c\_reload\_disable

The description of i2c\_reload\_disable is shown as below:

**Table 3-491. Function i2c\_reload\_disable**

<b>Function name</b>	i2c_reload_disable
<b>Function prototype</b>	void i2c_reload_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable I2C reload mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C reload mode */
```

```
i2c_reload_disable(I2C0);
```

### i2c\_transfer\_byte\_number\_config

The description of i2c\_transfer\_byte\_number\_config is shown as below:

**Table 3-492. Function i2c\_transfer\_byte\_number\_config**

<b>Function name</b>	i2c_transfer_byte_number_config
<b>Function prototype</b>	void i2c_transfer_byte_number_config(uint32_t i2c_periph, uint8_t byte_number);
<b>Function descriptions</b>	configure number of bytes to be transferred
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>byte_number</b>	0x0-0xFF, number of bytes to be transferred
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* configure number of bytes to be transferred */
```

```
i2c_transfer_byte_number_config(I2C0, 0xFF);
```

### i2c\_dma\_enable

The description of i2c\_dma\_enable is shown as below:

**Table 3-493. Function i2c\_dma\_enable**

<b>Function name</b>	i2c_dma_enable
<b>Function prototype</b>	void i2c_dma_enable(uint32_t i2c_periph, uint8_t dma);
<b>Function descriptions</b>	enable I2C DMA for transmission or reception
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>dma</b>	I2C DMA
<i>I2C_DMA_TRANSMIT</i>	transmit data using DMA
<i>I2C_DMA_RECEIVE</i>	receive data using DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C DMA for transmission or reception */
```

```
i2c_dma_enable(I2C0, I2C_DMA_RECEIVE);
```

### i2c\_dma\_disable

The description of i2c\_dma\_disable is shown as below:

**Table 3-494. Function i2c\_dma\_disable**

<b>Function name</b>	i2c_dma_disable
<b>Function prototype</b>	void i2c_dma_disable(uint32_t i2c_periph, uint8_t dma);
<b>Function descriptions</b>	disable I2C DMA for transmission or reception
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
<b>dma</b>	I2C DMA
<i>I2C_DMA_TRANSMIT</i>	transmit data using DMA
<i>I2C_DMA_RECEIVE</i>	receive data using DMA
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C DMA for transmission or reception */
```

```
i2c_dma_disable(I2C0, I2C_DMA_RECEIVE);
```

### i2c\_pec\_transfer

The description of i2c\_pec\_transfer is shown as below:

**Table 3-495. Function i2c\_pec\_transfer**

<b>Function name</b>	i2c_pec_transfer
<b>Function prototype</b>	void i2c_pec_transfer(uint32_t i2c_periph);
<b>Function descriptions</b>	I2C transfers PEC value
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
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 3-496. Function i2c\_pec\_enable**

<b>Function name</b>	i2c_pec_enable
----------------------	----------------

<b>Function prototype</b>	void i2c_pec_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable I2C PEC calculation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C PEC calculation */
i2c_pec_enable(I2C0);
```

## i2c\_pec\_disable

The description of i2c\_pec\_disable is shown as below:

**Table 3-497. Function i2c\_pec\_disable**

<b>Function name</b>	i2c_pec_disable
<b>Function prototype</b>	void i2c_pec_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable I2C PEC calculation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-498. Function i2c\_pec\_value\_get**

<b>Function name</b>	i2c_pec_value_get
----------------------	-------------------

<b>Function prototype</b>	uint32_t i2c_pec_value_get(uint32_t i2c_periph);
<b>Function descriptions</b>	get packet error checking value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	PEC value

Example:

```
/* I2C0 get packet error checking value */
uint32_t pec_value;
pec_value = i2c_pec_value_get(I2C0);
```

## i2c\_smbus\_alert\_enable

The description of i2c\_smbus\_alert\_enable is shown as below:

**Table 3-499. Function i2c\_smbus\_alert\_enable**

<b>Function name</b>	i2c_smbus_alert_enable
<b>Function prototype</b>	void i2c_smbus_alert_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable SMBus Alert
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SMBus Alert */
i2c_smbus_alert_enable(I2C0);
```

## i2c\_smbus\_alert\_disable

The description of i2c\_smbus\_alert\_disable is shown as below:

Table 3-500. 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=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SMBus Alert */
i2c_smbus_alert_disable(I2C0);
```

### i2c\_smbus\_default\_addr\_enable

The description of i2c\_smbus\_default\_addr\_enable is shown as below:

Table 3-501. 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=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SMBus device default address */
i2c_smbus_default_addr_enable(I2C0);
```

### i2c\_smbus\_default\_addr\_disable

The description of i2c\_smbus\_default\_addr\_disable is shown as below:

Table 3-502. 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=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SMBus device default address */
```

```
i2c_smbus_default_addr_disable(I2C0);
```

### i2c\_smbus\_host\_addr\_enable

The description of i2c\_smbus\_host\_addr\_enable is shown as below:

Table 3-503. 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=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SMBus Host address */
```

```
i2c_smbus_host_addr_enable(I2C0);
```

### i2c\_smbus\_host\_addr\_disable

The description of i2c\_smbus\_host\_addr\_disable is shown as below:



Table 3-504. 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=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SMBus Host address */
```

```
i2c_smbus_host_addr_disable(I2C0);
```

### i2c\_extented\_clock\_timeout\_enable

The description of i2c\_extented\_clock\_timeout\_enable is shown as below:

Table 3-505. 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=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_enable(I2C0);
```

### i2c\_extented\_clock\_timeout\_disable

The description of i2c\_extented\_clock\_timeout\_disable is shown as below:

Table 3-506. 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=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_disable(I2C0);
```

### i2c\_clock\_timeout\_enable

The description of i2c\_clock\_timeout\_enable is shown as below:

Table 3-507. 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=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable clock timeout detection */
```

```
i2c_clock_timeout_enable(I2C0);
```

### i2c\_clock\_timeout\_disable

The description of i2c\_clock\_timeout\_disable is shown as below:

**Table 3-508. Function i2c\_clock\_timeout\_disable**

<b>Function name</b>	i2c_clock_timeout_disable
<b>Function prototype</b>	void i2c_clock_timeout_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable clock timeout detection */
i2c_clock_timeout_disable(I2C0);
```

## i2c\_bus\_timeout\_b\_config

The description of i2c\_bus\_timeout\_b\_config is shown as below:

**Table 3-509. Function i2c\_bus\_timeout\_b\_config**

<b>Function name</b>	i2c_bus_timeout_b_config
<b>Function prototype</b>	void i2c_bus_timeout_b_config(uint32_t i2c_periph, uint32_t timeout);
<b>Function descriptions</b>	configure bus timeout B
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>timeout</b>	0-0xffff, bus timeout B
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure bus timeout B */
i2c_bus_timeout_b_config(I2C0, 0xff);
```

## i2c\_bus\_timeout\_a\_config

The description of i2c\_bus\_timeout\_a\_config is shown as below:

**Table 3-510. Function i2c\_bus\_timeout\_a\_config**

<b>Function name</b>	i2c_bus_timeout_a_config
<b>Function prototype</b>	void i2c_bus_timeout_a_config(uint32_t i2c_periph, uint32_t timeout);
<b>Function descriptions</b>	configure bus timeout A
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>timeout</b>	0-0xfff, bus timeout A
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure bus timeout A */
```

```
i2c_bus_timeout_a_config (I2C0, 0xff);
```

## i2c\_idle\_clock\_timeout\_config

The description of i2c\_idle\_clock\_timeout\_config is shown as below:

**Table 3-511. Function i2c\_idle\_clock\_timeout\_config**

<b>Function name</b>	i2c_idle_clock_timeout_config
<b>Function prototype</b>	void i2c_idle_clock_timeout_config(uint32_t i2c_periph, uint32_t timeout);
<b>Function descriptions</b>	configure idle clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>timeout</b>	bus timeout A
<i>BUSTOA_DETECT_SCL_LOW</i>	BUSTOA is used to detect SCL low timeout
<i>BUSTOA_DETECT_IDLE</i>	BUSTOA is used to detect both SCL and SDA high timeout when the bus is idle
<b>Output parameter{out}</b>	

-	-
<b>Return value</b>	
-	-

Example:

```
/* configure idle clock timeout detection */
```

```
i2c_idle_clock_timeout_config (I2C0, BUSTOA_DETECT_SCL_LOW);
```

## i2c\_flag\_get

The description of i2c\_flag\_get is shown as below:

**Table 3-512. Function i2c\_flag\_get**

<b>Function name</b>	i2c_flag_get
<b>Function prototype</b>	FlagStatus i2c_flag_get(uint32_t i2c_periph, uint32_t flag);
<b>Function descriptions</b>	get I2C flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
<b>flag</b>	I2C flags
I2C_FLAG_TBE	I2C_TDATA is empty during transmitting
I2C_FLAG_TI	transmit interrupt
I2C_FLAG_RBNE	I2C_RDATA is not empty during receiving
I2C_FLAG_ADDSEND	address received matches in slave mode
I2C_FLAG_NACK	not acknowledge flag
I2C_FLAG_STPDET	STOP condition detected in slave mode
I2C_FLAG_TC	transfer complete in master mode
I2C_FLAG_TCR	transfer complete reload
I2C_FLAG_BERR	bus error
I2C_FLAG_LOSTARB	arbitration Lost
I2C_FLAG_OUERR	overflow/underrun error in slave mode
I2C_FLAG_PECERR	PEC error
I2C_FLAG_TIMEOUT	timeout flag
I2C_FLAG_SMBALT	SMBus Alert
I2C_FLAG_I2CBSY	busy flag
I2C_FLAG_TR	whether the I2C is a transmitter or a receiver in slave mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get I2C flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get (I2C0, I2C_FLAG_TBE);
```

### i2c\_flag\_clear

The description of i2c\_flag\_clear is shown as below:

**Table 3-513. Function i2c\_flag\_clear**

<b>Function name</b>	i2c_flag_clear
<b>Function prototype</b>	void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);
<b>Function descriptions</b>	clear I2C flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
<b>flag</b>	I2C flags
I2C_FLAG_ADDSEND	address received matches in slave mode
I2C_FLAG_NACK	not acknowledge flag
I2C_FLAG_STPDET	STOP condition detected in slave mode
I2C_FLAG_BERR	bus error
I2C_FLAG_LOSTARB	arbitration Lost
I2C_FLAG_OUERR	overflow/underrun error in slave mode
I2C_FLAG_PECERR	PEC error
I2C_FLAG_TIMEOUT	timeout flag
I2C_FLAG_SMBALT	SMBus Alert
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-514. Function i2c\_interrupt\_enable**

<b>Function name</b>	i2c_interrupt_enable
<b>Function prototype</b>	void i2c_interrupt_enable(uint32_t i2c_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable I2C interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>interrupt</b>	I2C interrupts
<i>I2C_INT_ERR</i>	error interrupt
<i>I2C_INT_TC</i>	transfer complete interrupt
<i>I2C_INT_STPDET</i>	stop detection interrupt
<i>I2C_INT_NACK</i>	not acknowledge received interrupt
<i>I2C_INT_ADDM</i>	address match interrupt
<i>I2C_INT_RBNE</i>	receive interrupt
<i>I2C_INT_TI</i>	transmit interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 transmit interrupt */
i2c_interrupt_enable(I2C0, I2C_INT_TI);
```

## i2c\_interrupt\_disable

The description of i2c\_interrupt\_disable is shown as below:

**Table 3-515. Function i2c\_interrupt\_disable**

<b>Function name</b>	i2c_interrupt_disable
<b>Function prototype</b>	void i2c_interrupt_disable(uint32_t i2c_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable I2C interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>interrupt</b>	I2C interrupts
<i>I2C_INT_ERR</i>	error interrupt

<i>I2C_INT_TC</i>	transfer complete interrupt
<i>I2C_INT_STPDET</i>	stop detection interrupt
<i>I2C_INT_NACK</i>	not acknowledge received interrupt
<i>I2C_INT_ADDM</i>	address match interrupt
<i>I2C_INT_RBNE</i>	receive interrupt
<i>I2C_INT_TI</i>	transmit interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 transmit interrupt */
```

```
i2c_interrupt_disable(I2C0, I2C_INT_TI);
```

### i2c\_interrupt\_flag\_get

The description of i2c\_interrupt\_flag\_get is shown as below:

**Table 3-516. Function i2c\_interrupt\_flag\_get**

<b>Function name</b>	i2c_interrupt_flag_get
<b>Function prototype</b>	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	get I2C interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>int_flag</b>	I2C interrupt flags, refer to <a href="#">Table 3-454. i2c_interrupt_flag_enum</a> .
<i>I2C_INT_FLAG_TI</i>	transmit interrupt flag
<i>I2C_INT_FLAG_RBNE</i>	I2C_RDATA is not empty during receiving interrupt flag
<i>I2C_INT_FLAG_ADDS</i> <i>END</i>	address received matches in slave mode interrupt flag
<i>I2C_INT_FLAG_NACK</i>	not acknowledge interrupt flag
<i>I2C_INT_FLAG_STPD</i> <i>ET</i>	stop condition detected in slave mode interrupt flag
<i>I2C_INT_FLAG_TC</i>	transfer complete in master mode interrupt flag
<i>I2C_INT_FLAG_TCR</i>	transfer complete reload interrupt flag
<i>I2C_INT_FLAG_BERR</i>	bus error interrupt flag
<i>I2C_INT_FLAG_LOSTA</i> <i>RB</i>	arbitration lost interrupt flag



<i>I2C_INT_FLAG_OUER</i> <i>R</i>	overrun/underrun error in slave mode interrupt flag
<i>I2C_INT_FLAG_PECE</i> <i>RR</i>	PEC error interrupt flag
<i>I2C_INT_FLAG_TIMEO</i> <i>UT</i>	timeout interrupt flag
<i>I2C_INT_FLAG_SMBA</i> <i>LT</i>	SMBus Alert interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get I2C interrupt flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get(I2C0, I2C_INT_FLAG_TI);
```

### i2c\_interrupt\_flag\_clear

The description of i2c\_interrupt\_flag\_clear is shown as below:

**Table 3-517. Function i2c\_interrupt\_flag\_clear**

<b>Function name</b>	i2c_interrupt_flag_clear
<b>Function prototype</b>	void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	clear I2C interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>int_flag</b>	I2C interrupt flags, refer to <a href="#">Table 3-454. i2c_interrupt_flag_enum.</a>
<i>I2C_INT_FLAG_ADDS</i> <i>END</i>	address received matches in slave mode interrupt flag
<i>I2C_INT_FLAG_NACK</i>	not acknowledge interrupt flag
<i>I2C_INT_FLAG_STPD</i> <i>ET</i>	stop condition detected in slave mode interrupt flag
<i>I2C_INT_FLAG_BERR</i>	bus error interrupt flag
<i>I2C_INT_FLAG_LOSTA</i> <i>RB</i>	arbitration lost interrupt flag
<i>I2C_INT_FLAG_OUER</i>	overrun/underrun error in slave mode interrupt flag

<i>R</i>	
<i>I2C_INT_FLAG_PECERR</i>	PEC error interrupt flag
<i>I2C_INT_FLAG_TIMEOUT</i>	timeout interrupt flag
<i>I2C_INT_FLAG_SMBALERT</i>	SMBus Alert interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear a bus error flag */
```

```
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_BERR);
```

## 3.17. ICACHE

The instruction cache (ICACHE) is based on C-AHB code bus of Cortex-M33 processor. It is necessary to improve performance in fetching instruction and data from both internal and external memories. The ICACHE registers are listed in chapter [3.17.1](#), the ICACHE firmware functions are introduced in chapter [3.17.2](#).

### 3.17.1. Descriptions of Peripheral registers

ICACHE registers are listed in the table shown as below:

**Table 3-518. ICACHE Registers**

Registers	Descriptions
ICACHE_CTL	ICACHE control register
ICACHE_STAT	ICACHE status register
ICACHE_INTEN	ICACHE interrupt enable register
ICACHE_FC	ICACHE flag clear register
ICACHE_HMC	ICACHE hit monitor counter register
ICACHE_MMC	ICACHE miss monitor counter register
ICACHE_CFGx	ICACHE configuration register

### 3.17.2. Descriptions of Peripheral functions

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

**Table 3-519. ICACHE firmware function**

Function name	Function description
icache_enable	enable icache
icache_disable	disable icache
icache_monitor_enable	enable the icache monitor
icache_monitor_disable	disable the icache monitor
icache_monitor_reset	reset the icache monitor
icache_way_configure	configure icache way (associativity mode)
icache_burst_type_select	select icache burst type
icache_invalidation	invalidate icache
icache_hitvalue_get	get the hit monitor value
icache_missvalue_get	get the miss monitor value
icache_remap_enable	enable the icache remap function
icache_remap_disable	disable the icache remap function
icache_flag_get	get icache flag
icache_flag_clear	clear icache flag
icache_interrupt_enable	enable icache interrupt
icache_interrupt_disable	disable icache interrupt
icache_interrupt_flag_get	get icache interrupt flag
icache_interrupt_flag_clear	clear icache interrupt flag

## Structure icache\_remap\_struct

**Table 3-520. Structure icache\_remap\_struct**

Member name	Function description
region_num	remap region number
base_address	remap base address
remap_address	remap address
remap_size	remap size
burst_type	remap burst type
master_sel	remap master selection

## icache\_enable

The description of icache\_enable is shown as below:

**Table 3-521. Function icache\_enable**

Function name	icache_enable
Function prototype	void icache_enable(void);
Function descriptions	enable icache
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable icache */
```

```
icache_enable ();
```

## icache\_disable

The description of icache\_disable is shown as below:

**Table 3-522. Function icache\_disable**

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

Example:

```
/* disable icache */
```

```
icache_disable ();
```

## icache\_monitor\_enable

The description of icache\_monitor\_enable is shown as below:

**Table 3-523. Function icache\_monitor\_enable**

Function name	icache_monitor_enable
Function prototype	void icache_monitor_enable(uint32_t monitor_source);
Function descriptions	enable the icache monitor
Precondition	-
The called functions	-
Input parameter{in}	
monitor_source	the monitor to be enabled
ICACHE_MONITOR_HIT	hit monitor

ICACHE_MONITOR_MISS	miss monitor
ICACHE_MONITOR_HIT_MISS	hit and miss monitor
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the icache monitor */
```

```
icache_monitor_enable(ICACHE_MONITOR_HIT);
```

## icache\_monitor\_disable

The description of icache\_monitor\_disable is shown as below:

**Table 3-524. Function icache\_monitor\_disable**

<b>Function name</b>	icache_monitor_disable
<b>Function prototype</b>	void icache_monitor_disable(uint32_t monitor_source);
<b>Function descriptions</b>	disable the icache monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>monitor_source</b>	the monitor to be disabled
ICACHE_MONITOR_HIT	hit monitor
ICACHE_MONITOR_MISS	miss monitor
ICACHE_MONITOR_HIT_MISS	hit and miss monitor
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the icache monitor */
```

```
icache_monitor_disable(ICACHE_MONITOR_HIT);
```

## icache\_monitor\_reset

The description of icache\_monitor\_reset is shown as below:

**Table 3-525. Function icache\_monitor\_reset**

<b>Function name</b>	icache_monitor_reset
<b>Function prototype</b>	void icache_monitor_reset(uint32_t reset_monitor_source);
<b>Function descriptions</b>	reset the icache monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>reset_monitor_source</b>	the monitor to be reset
ICACHE_MONITOR_HIT	hit monitor
ICACHE_MONITOR_MISS	miss monitor
ICACHE_MONITOR_HIT_MISS	hit and miss monitor
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset the icache monitor */
```

```
icache_monitor_reset (ICACHE_MONITOR_HIT);
```

## icache\_way\_configure

The description of icache\_way\_configure is shown as below:

**Table 3-526. Function icache\_way\_configure**

<b>Function name</b>	icache_way_configure
<b>Function prototype</b>	ErrStatus icache_way_configure(void);
<b>Function descriptions</b>	Configure icache way (associativity mode)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* Configure icache way */
```

```
ErrStatus Flag = icache_way_configure ();
```

## icache\_burst\_type\_select

The description of icache\_burst\_type\_select is shown as below:

**Table 3-527. Function icache\_burst\_type\_select**

<b>Function name</b>	icache_burst_type_select
<b>Function prototype</b>	ErrStatus icache_burst_type_select(uint32_t burst_type);
<b>Function descriptions</b>	select icache burst type
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>burst_type</b>	output burst type
ICACHE_WRAP_BURST	icache WRAP burst mode
ICACHE_INCR_BURST	icache INCR burst mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* select icache burst type */
```

```
ErrStatus Flag = icache_burst_type_select (ICACHE_WRAP_BURST);
```

## icache\_invalidation

The description of icache\_invalidation is shown as below:

**Table 3-528. Function icache\_invalidation**

<b>Function name</b>	icache_invalidation
<b>Function prototype</b>	ErrStatus icache_invalidation(void);
<b>Function descriptions</b>	invalidate icache
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* invalid icache */
```

ErrStatus Flag = icache\_invalidation();

## icache\_hitvalue\_get

The description of icache\_hitvalue\_get is shown as below:

**Table 3-529. Function icache\_hitvalue\_get**

<b>Function name</b>	icache_hitvalue_get
<b>Function prototype</b>	uint32_t icache_hitvalue_get(void);
<b>Function descriptions</b>	get the hit monitor value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	32-bit value of the hit monitor counter register (0-0Xffff ffff)

Example:

```
/* get the hit monitor value */
uint8_t hit_value = 0;
hit_value = icache_hitvalue_get ();
```

## icache\_missvalue\_get

The description of icache\_missvalue\_get is shown as below:

**Table 3-530. Function icache\_missvalue\_get**

<b>Function name</b>	icache_missvalue_get
<b>Function prototype</b>	uint32_t icache_missvalue_get(void);
<b>Function descriptions</b>	get the miss monitor value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	lower 16-bit value of the miss monitor counter register (0-0Xffff)

Example:

```
/* get the miss monitor value */
```



```
uint8_t miss_value = 0;
```

```
miss_value = icache_missvalue_get ();
```

## icache\_remap\_enable

The description of icache\_remap\_enable is shown as below:

**Table 3-531. Function icache\_remap\_enable**

<b>Function name</b>	icache_remap_enable
<b>Function prototype</b>	ErrStatus icache_remap_enable(icache_remap_struct * icache_remap_config);
<b>Function descriptions</b>	enable the icache remap function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
icache_remap_config	icache remap structure, refer to <a href="#">Table 3-520. Structure icache_remap_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
ErrStatus	SUCCESS or ERROR

Example:

```
/* enable the icache remap function */

icache_remap_struct ICACHE_struct;

ICACHE_struct.region_num = 0U;

ICACHE_struct.base_address = 0x10000000UL;

ICACHE_struct.remap_address = 0x30000000UL;

ICACHE_struct.remap_size = ICACHE_REMAP_2M;

ICACHE_struct.burst_type = ICACHE_WRAP_BURST;

icache_remap_enable(&ICACHE_struct);
```

## icache\_remap\_disable

The description of icache\_remap\_disable is shown as below:

**Table 3-532. Function icache\_remap\_disable**

<b>Function name</b>	icache_remap_disable
<b>Function prototype</b>	ErrStatus icache_remap_disable(uint32_t region_num);
<b>Function descriptions</b>	disable the icache remap function
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>region_num</b>	the remap region to be disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* disable the icache remap function */
```

```
icache_remap_disable (1);
```

## icache\_flag\_get

The description of icache\_flag\_get is shown as below:

**Table 3-533. Function icache\_flag\_get**

<b>Function name</b>	icache_flag_get
<b>Function prototype</b>	FlagStatus icache_flag_get(uint32_t flag);
<b>Function descriptions</b>	get icache flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	the icache flag to be get
ICACHE_BUSY_FLAG	icache busy flag
ICACHE_END_FLAG	icache operation end flag
ICACHE_ERR_FLAG	icache error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get icache flag */
```

```
FlagStatus flag_value;
```

```
flag_value = icache_flag_get (ICACHE_BUSY_FLAG);
```

## icache\_flag\_clear

The description of icache\_flag\_clear is shown as below:

**Table 3-534. Function icache\_flag\_clear**

<b>Function name</b>	icache_flag_clear
----------------------	-------------------

<b>Function prototype</b>	void icache_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear icache flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	the flag to be cleared
<i>ICACHE_ENDC_FLAG</i>	icache operation end clear flag
<i>ICACHE_ERRC_FLAG</i>	icache error clear flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear icache flag */
```

```
icache_flag_clear (ICACHE_ENDC_FLAG);
```

## icache\_interrupt\_enable

The description of icache\_interrupt\_enable is shown as below:

**Table 3-535. Function icache\_interrupt\_enable**

<b>Function name</b>	icache_interrupt_enable
<b>Function prototype</b>	void icache_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable icache interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the interrupt to be enabled
<i>ICACHE_ENDIE</i> :	icache operation end interrupt
<i>ICACHE_ERRIE</i>	icache error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable icache interrupt */
```

```
icache_interrupt_enable (ICACHE_ENDIE);
```

## icache\_interrupt\_disable

The description of icache\_interrupt\_disable is shown as below:

Table 3-536. Function `icache_interrupt_disable`

Function name	<code>icache_interrupt_disable</code>
Function prototype	<code>void icache_interrupt_disable(uint32_t interrupt);</code>
Function descriptions	disable icache interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	the interrupt to be disabled
<code>ICACHE_ENDIE</code>	icache operation end interrupt
<code>ICACHE_ERRIE</code>	icache error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable icache interrupt */
```

```
icache_interrupt_disable (ICACHE_ENDIE);
```

### `icache_interrupt_flag_get`

The description of `icache_interrupt_flag_get` is shown as below:

Table 3-537. Function `icache_interrupt_flag_get`

Function name	<code>icache_interrupt_flag_get</code>
Function prototype	<code>FlagStatus icache_interrupt_flag_get(uint32_t interrupt);</code>
Function descriptions	get icache interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	the icache interrupt flag to be get
<code>ICACHE_END_FLAG</code>	icache operation end interrupt flag
<code>ICACHE_ERR_FLAG</code>	icache error interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get icache flag */
```

```
FlagStatus flag_value;
```

```
flag_value = icache_interrupt_flag_get (ICACHE_END_FLAG);
```

## icache\_interrupt\_flag\_clear

The description of icache\_interrupt\_flag\_clear is shown as below:

**Table 3-538. Function icache\_interrupt\_flag\_clear**

<b>Function name</b>	icache_interrupt_flag_clear
<b>Function prototype</b>	void icache_interrupt_flag_clear(uint32_t interrupt);
<b>Function descriptions</b>	clear icache interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the interrupt flag to clear
ICACHE_ENDC_FLAG	icache operation end clear interrupt flag
ICACHE_ERRC_FLAG	icache error clear interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear icache interrupt flag */
```

```
icache_interrupt_flag_clear (ICACHE_ENDC_FLAG);
```

## 3.18. MISC

MISC is a software package that provide the interfaces for NVIC and SysTick. The NVIC and SysTick registers are listed in chapter [3.18.1](#), the MISC firmware functions are introduced in chapter [3.18.2](#).

### 3.18.1. Descriptions of Peripheral registers

**Table 3-539. NVIC Registers**

Registers	Descriptions
ISER <sup>(1)</sup>	Interrupt Set Enable Register
ICER <sup>(1)</sup>	Interrupt Clear Enable Register
ISPR <sup>(1)</sup>	Interrupt Set Pending Register
ICPR <sup>(1)</sup>	Interrupt Clear Pending Register
IABR <sup>(1)</sup>	Interrupt Active bit Register
ITNS <sup>(1)</sup>	Interrupt Non-Secure State Register
IPR <sup>(1)</sup>	Interrupt Priority Register
STIR <sup>(1)</sup>	Software Trigger Interrupt Register
CPUID <sup>(2)</sup>	CPUID Base Register

Registers	Descriptions
ICSR <sup>(2)</sup>	Interrupt Control and State Register
VTOR <sup>(2)</sup>	Vector Table Offset Register
AIRCR <sup>(2)</sup>	Application Interrupt and Reset Control Register
SCR <sup>(2)</sup>	System Control Register
CCR <sup>(2)</sup>	Configuration Control Register
SHPR <sup>(2)</sup>	System Handlers Priority Registers
SHCSR <sup>(2)</sup>	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 3-540. SysTick Registers**

Registers	Descriptions
CTRL <sup>(1)</sup>	SysTick Control and Status Register
LOAD <sup>(1)</sup>	SysTick Reload Value Register
VAL <sup>(1)</sup>	SysTick Current Value Register
CALIB <sup>(1)</sup>	SysTick Calibration Register

1. refer to the structure SysTick\_Type, is defined in the core\_cm33.h file

### 3.18.2. Descriptions of Peripheral functions

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

**Table 3-541. 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 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 3-542. IRQn\_Type ()**

Member name	Function description
WWDGT_IRQn	Window watchdog timer interrupt
LVD_IRQn	LVD through EXTI Line detection interrupt
TAMPER_STAMP_IRQn	RTC Tamper and TimeStamp events interrupt
RTC_WKUP_IRQn	RTC Wakeup event interrupt
FMC_IRQn	FMC global interrupt
RCU_IRQn	RCU global interrupt

Member name	Function description
EXTI0_IRQn	EXTI Line0 interrupt
EXTI1_IRQn	EXTI Line1 interrupt
EXTI2_IRQn	EXTI Line2 interrupt
EXTI3_IRQn	EXTI Line3 interrupt
EXTI4_IRQn	EXTI Line4 interrupt
DMA0_Channel0_IRQn	DMA0 channel0 global interrupt
DMA0_Channel1_IRQn	DMA0 channel1 global interrupt
DMA0_Channel2_IRQn	DMA0 channel2 global interrupt
DMA0_Channel3_IRQn	DMA0 channel3 global interrupt
DMA0_Channel4_IRQn	DMA0 channel4 global interrupt
DMA0_Channel5_IRQn	DMA0 channel5 global interrupt
DMA0_Channel6_IRQn	DMA0 channel6 global interrupt
DMA0_Channel7_IRQn	DMA0 channel7 global interrupt
ADC_IRQn	ADC interrupt
TAMPER_STAMP_S_IRQn	RTC Tamper and TimeStamp events security interrupt
RTC_WKUP_S_IRQn	RTC Wakeup event security interrupt
RTC_Alarm_S_IRQn	RTC Alarm event security interrupt
EXTI5_9_IRQn	EXTI Line5-9 interrupt
TIMER0_BRK_IRQn	TIMER0 Break interrupt
TIMER0_UP_IRQn	TIMER0 update interrupt
TIMER0_CMT_IRQn	TIMER0 commutation interrupt
TIMER0_Channel_IRQn	TIMER0 Capture Compare interrupt
TIMER1_IRQn	TIMER1 global interrupt
TIMER2_IRQn	TIMER2 global interrupt
TIMER3_IRQn	TIMER3 global interrupt
I2C0_EV_IRQn	I2C0 event interrupt
I2C0_ER_IRQn	I2C0 error interrupt
I2C1_EV_IRQn	I2C1 event interrupt
I2C1_ER_IRQn	I2C1 error interrupt
SPI0_IRQn	SPI0 global interrupt
SPI1_IRQn	SPI1/I2S1 global interrupt
USART0_IRQn	USART0 global interrupt
USART1_IRQn	USART1 global interrupt
USART2_IRQn	USART2 global interrupt
EXTI10_15_IRQn	EXTI Line10-15 interrupt
RTC_Alarm_IRQn	RTC Alarm event interrupt
VLVDF_IRQn	VLVDF interrupt
TIMER15_IRQn	TIMER15 global interrupt
TIMER16_IRQn	TIMER16 global interrupt
SDIO_IRQn	SDIO global interrupt
TIMER4_IRQn	TIMER4 global interrupt

Member name	Function description
I2C0_WKUP_IRQn	I2C0 Wakeup interrupt
USART0_IRQn	USART0 Wakeup
USART2_IRQn	USART2 Wakeup
TIMER5_IRQn	TIMER5 global interrupt
DMA1_Channel0_IRQn	DMA1 channel0 global interrupt
DMA1_Channel1_IRQn	DMA1 channel1 global interrupt
DMA1_Channel2_IRQn	DMA1 channel2 global interrupt
DMA1_Channel3_IRQn	DMA1 channel3 global interrupt
DMA1_Channel4_IRQn	DMA1 channel4 global interrupt
DMA1_Channel5_IRQn	DMA1 channel5 global interrupt
DMA1_Channel6_IRQn	DMA1 channel6 global interrupt
DMA1_Channel7_IRQn	DMA1 channel7 global interrupt
WIFI11N_WKUP_IRQn	WIFI11N wakeup interrupt
USBFS_IRQn	USBFS global interrupt
USBFS_WKUP_IRQn	USBFS wakeup interrupt
DCI_IRQn	DCI global interrupt
CAU_IRQn	CAU global interrupt
HAU_TRNG_IRQn	HAU/TRNG global interrupt
FPU_IRQn	FPU global interrupt
HPDF_INT0_IRQn	HPDF global interrupt0
HPDF_INT1_IRQn	HPDF global interrupt1
WIFI11N_INT0_IRQn	WIFI11N global interrupt0
WIFI11N_INT1_IRQn	WIFI11N global interrupt1
WIFI11N_INT2_IRQn	WIFI11N global interrupt2
EFUSE_IRQn	EFUSE global interrupt
QSPI_IRQn	QSPI global interrupt
PKCAU_IRQn	PKCAU global interrupt
TSI_IRQ	TSI global interrupt
ICACHE_IRQn	ICACHE global interrupt
TZIAC_S_IRQn	TZIAC security interrupt
FMC_S_IRQn	FMC secure interrupt
QSPI_S_IRQn	QSPI security interrupt

## nvic\_priority\_group\_set

The description of nvic\_priority\_group\_set is shown as below:

**Table 3-543. Function nvic\_priority\_group\_set**

Function name	nvic_priority_group_set
Function prototype	void nvic_priority_group_set(uint32_t nvic_prigroup);
Function descriptions	configure bits length of the priority group
Precondition	-



<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>nvic_prigroup</b>	priority group
<i>NVIC_PRIGROUP_PRE0_SUB4</i>	0 bits for pre-emption priority 4 bits for subpriority
<i>NVIC_PRIGROUP_PRE1_SUB3</i>	1 bits for pre-emption priority 3 bits for subpriority
<i>NVIC_PRIGROUP_PRE2_SUB2</i>	2 bits for pre-emption priority 2 bits for subpriority
<i>NVIC_PRIGROUP_PRE3_SUB1</i>	3 bits for pre-emption priority 1 bits for subpriority
<i>NVIC_PRIGROUP_PRE4_SUB0</i>	4 bits for pre-emption priority 0 bits for subpriority
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-544. Function nvic\_irq\_enable**

<b>Function name</b>	nvic_irq_enable
<b>Function prototype</b>	void nvic_irq_enable(IRQn_Type nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
<b>Function descriptions</b>	enable NVIC request, configure the priority of interrupt
<b>Precondition</b>	-
<b>The called functions</b>	nvic_priority_group_set
<b>Input parameter{in}</b>	
<b>nvic_irq</b>	NVIC interrupt, refer to <a href="#">Table 3-542. IRQn_Type</a>
<b>Input parameter{in}</b>	
<b>nvic_irq_pre_priority</b>	the pre-emption priority needed to set (0~4)
<b>Input parameter{in}</b>	
<b>nvic_irq_sub_priority</b>	the subpriority needed to set (0~4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable window watchDog timer interrupt , pre-emption priority is 1, subpriority is 1 */
nvc_irq_enable(WWDGT_IRQn, 1, 1);
```

## nvc\_irq\_disable

The description of nvc\_irq\_disable is shown as below:

**Table 3-545. Function nvc\_irq\_disable**

<b>Function name</b>	nvc_irq_disable
<b>Function prototype</b>	void nvc_irq_disable(IRQn_Type nvc_irq);
<b>Function descriptions</b>	disable NVIC request
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
nvc_irq	NVIC interrupt, refer to <a href="#">Table 3-542. IRQn_Type</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable window watchDog timer interrupt */
nvc_irq_disable(WWDGT_IRQn);
```

## nvc\_system\_reset

The description of nvc\_system\_reset is shown as below:

**Table 3-546. Function nvc\_system\_reset**

<b>Function name</b>	nvc_system_reset
<b>Function prototype</b>	void nvc_system_reset(void);
<b>Function descriptions</b>	initiates a system reset request to reset the MCU
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-547. Function `nvic_vector_table_set`**

<b>Function name</b>	<code>nvic_vector_table_set</code>
<b>Function prototype</b>	<code>void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);</code>
<b>Function descriptions</b>	set the NVIC vector table address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>nvic_vect_tab</b>	the RAM or FLASH base address
<code>NVIC_VECTTAB_RAM</code>	RAM base address
<code>NVIC_VECTTAB_FLASH</code>	Flash base address
<code>NVIC_VECTTAB_RAM_SECURE</code>	RAM base address of secure world
<code>NVIC_VECTTAB_FLASH_SECURE</code>	Flash base address of secure world
<b>Input parameter{in}</b>	
<b>offset</b>	Vector Table offset (vector table start address= base address+offset)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-548. Function `system_lowpower_set`**

<b>Function name</b>	<code>system_lowpower_set</code>
<b>Function prototype</b>	<code>void system_lowpower_set(uint8_t lowpower_mode);</code>
<b>Function descriptions</b>	the state of the low power mode management
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowpower_mode</b>	the low power mode state

<i>SCB_LPM_SLEEP_EXIT_ISR</i>	if chose this para, the system always enter low power mode by exiting from ISR
<i>SCB_LPM_DEEPSLEEP</i>	if chose this para, the system will enter the DEEPSLEEP mode
<i>SCB_LPM_WAKE_BY_ALL_INT</i>	if chose this para, the lowpower mode can be woke up by all the enable and disable interrupts
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-549. Function system\_lowpower\_reset**

<b>Function name</b>	system_lowpower_reset
<b>Function prototype</b>	void system_lowpower_reset(uint8_t lowpower_mode);
<b>Function descriptions</b>	the state of the low power mode management
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowpower_mode</b>	the low power mode state
<i>SCB_LPM_SLEEP_EXIT_ISR</i>	if chose this para, the system will exit low power mode by exiting from ISR
<i>SCB_LPM_DEEPSLEEP</i>	if chose this para, the system will enter the SLEEP mode
<i>SCB_LPM_WAKE_BY_ALL_INT</i>	if chose this para, the lowpower mode only can be woke up by the enable interrupts
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-550. Function systick\_clksource\_set**

<b>Function name</b>	systick_clksource_set
<b>Function prototype</b>	void systick_clksource_set(uint32_t systick_clksource);
<b>Function descriptions</b>	set the systick clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>systick_clksource</b>	the systick clock source needed to choose
SYSTICK_CLKSOURC E_HCLK	systick clock source is from HCLK
SYSTICK_CLKSOURC E_HCLK_DIV8	systick clock source is from HCLK/8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* systick clock source is HCLK/8 */
```

```
systick_clksource_set (SYSTICK_CLKSOURCE_HCLK_DIV8);
```

## 3.19. PKCAU

The Public Key Cryptographic Acceleration Unit (PKCAU) can accelerate RSA (Rivest, Shamir and Adleman), Diffie-Hellmann (DH key exchange) and ECC (elliptic curve cryptography) in GF(p) (Galois domain). The PKCAU registers are listed in chapter [3.19.1](#), the PKCAU firmware functions are introduced in chapter [3.19.2](#).

### 3.19.1. Descriptions of Peripheral registers

PKCAU registers are listed in the table shown as below:

**Table 3-551. PKCAU Registers**

Registers	Descriptions
PKCAU_CTL	Control register
PKCAU_STAT	Status register
PKCAU_STATC	Status clear register

### 3.19.2. Descriptions of Peripheral functions

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

**Table 3-552. PKCAU firmware function**

Function name	Function description
pkcau_deinit	reset PKCAU
pkcau_mont_struct_para_init	initialize montgomery parameter structure with a default value
pkcau_mod_struct_para_init	initialize modular parameter structure with a default value
pkcau_mod_exp_struct_para_init	initialize modular exponentation parameter structure with a default value
pkcau_mod_inver_struct_para_init	initialize modular inversion parameter structure with a default value
pkcau_arithmetic_struct_para_init	initialize arithmetic parameter structure with a default value
pkcau_crt_struct_para_init	initialize CRT parameter structure with a default value
pkcau_ec_group_struct_para_init	initialize ECC curve parameter structure with a default value
pkcau_point_struct_para_init	initialize point parameter structure with a default value
pkcau_signature_struct_para_init	initialize signature parameter structure with a default value
pkcau_hash_struct_para_init	initialize hash parameter structure with a default value
pkcau_mod_reduc_struct_para_init	initialize modular reduction parameter structure with a default value
pkcau_enable	enable PKCAU
pkcau_disable	disable PKCAU
pkcau_start	start operation
pkcau_mode_set	configure the PKCAU operation mode
pkcau_mont_param_operation	execute montgomery parameter operation
pkcau_mod_operation	execute modular operation, include modular addition, modular subtraction and montgomery multiplication
pkcau_mod_exp_operation	execute modular exponentation operation
pkcau_mod_inver_operation	execute modular inversion operation
pkcau_mod_reduc_operation	execute modular reduction operation
pkcau_arithmetic_operation	execute arithmetic addition operation
pkcau_crt_exp_operation	execute RSA CRT exponentation operation
pkcau_point_check_operation	execute point check operation
pkcau_point_mul_operation	execute point multiplication operation
pkcau_ecdsa_sign_operation	execute ECDSA sign operation
pkcau_ecdsa_verification_operation	execute ECDSA verify operation
pkcau_memread	read result from PKCAU RAM
pkcau_flag_get	get PKCAU flag status
pkcau_flag_clear	clear PKCAU flag status
pkcau_interrupt_enable	enable PKCAU interrupt
pkcau_interrupt_disable	disable PKCAU interrupt
pkcau_interrupt_flag_get	get PKCAU interrupt flag status

Function name	Function description
pkcau_interrupt_flag_clear	clear PKCAU interrupt flag status

## Structure pkcau\_mont\_parameter\_struct

**Table 3-553. Structure pkcau\_mont\_parameter\_struct**

Member name	Function description
modulus_n	modulus value n
modulus_len	modulus length in byte

## Structure pkcau\_mod\_parameter\_struct

**Table 3-554. Structure pkcau\_mod\_parameter\_struct**

Member name	Function description
opr_d_a	operand A
opr_d_b	operand B
modulus_n	modulus value n
modulus_len	modulus length in byte

## Structure pkcau\_mod\_exp\_parameter\_struct

**Table 3-555. Structure pkcau\_mod\_exp\_parameter\_struct**

Member name	Function description
opr_d_a	operand A
exp_e	exponent e
e_len	exponent length in byte
modulus_n	modulus n
modulus_len	modulus length in byte
mont_para	montgomery parameter R2 mod n

## Structure pkcau\_mod\_inver\_parameter\_struct

**Table 3-556. Structure pkcau\_mod\_inver\_parameter\_struct**

Member name	Function description
opr_d_a	operand A
modulus_n	modulus value n
modulus_len	modulus length in byte

## Structure pkcau\_mod\_reduc\_parameter\_struct

**Table 3-557. Structure pkcau\_mod\_reduc\_parameter\_struct**

Member name	Function description
opr_d_a	operand A
opr_d_a_len	length of operand A in byte
modulus_n	modulus value n

Member name	Function description
modulus_len	modulus length in byte

## Structure pkcau\_arithmetic\_parameter\_struct

**Table 3-558. Structure pkcau\_arithmetic\_parameter\_struct**

Member name	Function description
opr_d_a	operand A
opr_d_b	operand B
opr_d_len	length of operand in byte

## Structure pkcau\_crt\_parameter\_struct

**Table 3-559. Structure pkcau\_crt\_parameter\_struct**

Member name	Function description
opr_d_a	operand A
opr_d_len	length of operand in byte
opr_d_dp	operand dp
opr_d_dq	operand dq
opr_d_qinv	operand qinv
opr_d_p	prime operand p
opr_d_q	prime operand q

## Structure pkcau\_ec\_group\_parameter\_struct

**Table 3-560. Structure pkcau\_ec\_group\_parameter\_struct**

Member name	Function description
modulus_p	curve modulus p
coff_a	curve coefficient a
coff_b	curve coefficient b
base_point_x	curve base point coordinate x
base_point_y	curve base point coordinate y
order_n	curve prime order n
a_sign	curve coefficient a sign
modulus_p_len	curve modulus p length in byte
order_n_len	curve prime order n length in byte
multi_k	scalar multiplier k
k_len	length of scalar multiplier k
mont_para	montgomery parameter R2 mod n



## Structure pkcau\_point\_parameter\_struct

**Table 3-561. Structure pkcau\_point\_parameter\_struct**

Member name	Function description
point_x	point coordinate x
point_y	point coordinate y

## Structure pkcau\_signature\_parameter\_struct

**Table 3-562. Structure pkcau\_signature\_parameter\_struct**

Member name	Function description
sign_r	signature part r
sign_s	signature part s

## Structure pkcau\_hash\_parameter\_struct

**Table 3-563. Structure pkcau\_hash\_parameter\_struct**

Member name	Function description
hash_z	hash value z
hash_z_len	hash value z length in byte

## pkcau\_deinit

The description of pkcau\_deinit is shown as below:

**Table 3-564. Function pkcau\_deinit**

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

Example:

```
/* reset PKCAU */
pkcau_deinit();
```

## pkcau\_mont\_struct\_para\_init

The description of pkcau\_mont\_struct\_para\_init is shown as below:

Table 3-565. Function pkcau\_mont\_struct\_para\_init

Function name	pkcau_mont_struct_para_init
Function prototype	void pkcau_mont_struct_para_init(pkcau_mont_parameter_struct* init_para);
Function descriptions	initialize montgomery parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	montgomery parameter struct, the structure members can refer to <a href="#">Table 3-553. Structure pkcau_mont_parameter_struct.</a>
Return value	
-	-

Example:

```
/* initialize PKCAU montgomery parameter struct with a default value */
```

```
pkcau_mont_parameter_struct pkcau_mont_parameter;
```

```
pkcau_mont_struct_para_init(&pkcau_mont_parameter);
```

### pkcau\_mod\_struct\_para\_init

The description of pkcau\_mod\_struct\_para\_init is shown as below:

Table 3-566. Function pkcau\_mod\_struct\_para\_init

Function name	pkcau_mod_struct_para_init
Function prototype	void pkcau_mod_struct_para_init(pkcau_mod_parameter_struct* init_para);
Function descriptions	initialize modular parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	modular parameter struct, the structure members can refer to <a href="#">Table 3-554. Structure pkcau_mod_parameter_struct.</a>
Return value	
-	-

Example:

```
/* initialize PKCAU modular parameter struct with a default value */
```

```
pkcau_mod_parameter_struct pkcau_mod_parameter;
```

```
pkcau_mod_struct_para_init(&pkcau_mod_parameter);
```

## pkcau\_mod\_exp\_struct\_para\_init

The description of pkcau\_mod\_exp\_struct\_para\_init is shown as below:

**Table 3-567. Function pkcau\_mod\_exp\_struct\_para\_init**

<b>Function name</b>	pkcau_mod_exp_struct_para_init
<b>Function prototype</b>	void pkcau_mod_exp_struct_para_init(pkcau_mod_exp_parameter_struct* init_para);
<b>Function descriptions</b>	initialize modular exponentation parameter structure with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
init_para	modular exponentation parameter struct, the structure members can refer to <a href="#">Table 3-555. Structure pkcau_mod_exp_parameter_struct.</a>
<b>Return value</b>	
-	-

Example:

```
/* initialize PKCAU modular exponentation parameter struct with a default value */
```

```
pkcau_mod_exp_parameter_struct pkcau_mod_exp_parameter;
```

```
pkcau_mod_exp_struct_para_init(&pkcau_mod_exp_parameter);
```

## pkcau\_mod\_inver\_struct\_para\_init

The description of pkcau\_mod\_inver\_struct\_para\_init is shown as below:

**Table 3-568. Function pkcau\_mod\_inver\_struct\_para\_init**

<b>Function name</b>	pkcau_mod_inver_struct_para_init
<b>Function prototype</b>	void pkcau_mod_inver_struct_para_init(pkcau_mod_inver_parameter_struct* init_para);
<b>Function descriptions</b>	initialize modular inversion parameter structure with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
init_para	modular inversion parameter struct, the structure members can refer to <a href="#">Table 3-556. Structure pkcau_mod_inver_parameter_struct.</a>
<b>Return value</b>	
-	-

Example:

```
/* initialize PKCAU modular inversion parameter struct with a default value */

pkcau_mod_inver_parameter_struct pkcau_mod_inver_parameter;

pkcau_mod_inver_struct_para_init(&pkcau_mod_inver_parameter);
```

### pkcau\_arithmetic\_struct\_para\_init

The description of pkcau\_arithmetic\_struct\_para\_init is shown as below:

**Table 3-569. Function pkcau\_arithmetic\_struct\_para\_init**

<b>Function name</b>	pkcau_arithmetic_struct_para_init
<b>Function prototype</b>	void pkcau_arithmetic_struct_para_init(pkcau_arithmetic_parameter_struct* init_para);
<b>Function descriptions</b>	initialize arithmetic parameter structure with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
init_para	arithmetic parameter struct, the structure members can refer to <a href="#">Table 3-558. Structure pkcau_arithmetic_parameter_struct</a> .
<b>Return value</b>	
-	-

Example:

```
/* initialize PKCAU arithmetic parameter struct struct with a default value */

pkcau_arithmetic_parameter_struct pkcau_arithmetic_parameter;

pkcau_arithmetic_struct_para_init(&pkcau_arithmetic_parameter);
```

### pkcau\_crt\_struct\_para\_init

The description of pkcau\_crt\_struct\_para\_init is shown as below:

**Table 3-570. Function pkcau\_crt\_struct\_para\_init**

<b>Function name</b>	pkcau_crt_struct_para_init
<b>Function prototype</b>	void pkcau_crt_struct_para_init(pkcau_crt_parameter_struct* init_para);
<b>Function descriptions</b>	initialize CRT parameter structure with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	

<b>init_para</b>	CRT parameter struct, the structure members can refer to <a href="#">Table 3-559. Structure pkcau crt parameter struct.</a>
<b>Return value</b>	
-	-

Example:

```
/* initialize PKCAU CRT parameter struct with a default value */
```

```
pkcau_crt_parameter_struct pkcau_crt_parameter;
```

```
pkcau_crt_struct_para_init(&pkcau_crt_parameter);
```

### pkcau\_ec\_group\_struct\_para\_init

The description of pkcau\_ec\_group\_struct\_para\_init is shown as below:

**Table 3-571. Function pkcau\_ec\_group\_struct\_para\_init**

<b>Function name</b>	pkcau_ec_group_struct_para_init
<b>Function prototype</b>	void pkcau_ec_group_struct_para_init(pkcau_ec_group_parameter_struct* init_para);
<b>Function descriptions</b>	initialize ECC curve parameter structure with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>init_para</b>	ECC curve parameter struct, the structure members can refer to <a href="#">Table 3-560. Structure pkcau ec group parameter struct.</a>
<b>Return value</b>	
-	-

Example:

```
/* initialize PKCAU ECC curve parameter struct with a default value */
```

```
pkcau_ec_group_parameter_struct pkcau_ec_group_parameter;
```

```
pkcau_ec_group_struct_para_init(&pkcau_ec_group_parameter);
```

### pkcau\_point\_struct\_para\_init

The description of pkcau\_point\_struct\_para\_init is shown as below:

**Table 3-572. Function pkcau\_point\_struct\_para\_init**

<b>Function name</b>	pkcau_point_struct_para_init
<b>Function prototype</b>	void pkcau_point_struct_para_init(pkcau_point_parameter_struct* init_para);
<b>Function descriptions</b>	initialize point parameter structure with a default value

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>init_para</b>	point parameter struct, the structure members can refer to <a href="#">Table 3-561. Structure pkcau_point_parameter_struct.</a>
<b>Return value</b>	
-	-

Example:

```
/* initialize PKCAU point parameter struct with a default value */
pkcau_point_parameter_struct pkcau_point_parameter;
pkcau_point_struct_para_init(&pkcau_point_parameter);
```

### pkcau\_signature\_struct\_para\_init

The description of pkcau\_signature\_struct\_para\_init is shown as below:

**Table 3-573. Function pkcau\_signature\_struct\_para\_init**

<b>Function name</b>	pkcau_signature_struct_para_init
<b>Function prototype</b>	void pkcau_signature_struct_para_init(pkcau_signature_parameter_struct* init_para);
<b>Function descriptions</b>	initialize signature parameter structure with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>init_para</b>	signature parameter struct, the structure members can refer to <a href="#">Table 3-562. Structure pkcau_signature_parameter_struct.</a>
<b>Return value</b>	
-	-

Example:

```
/* initialize PKCAU signature parameter struct with a default value */
pkcau_signature_parameter_struct pkcau_signature_parameter;
pkcau_signature_struct_para_init(&pkcau_signature_parameter);
```

### pkcau\_hash\_struct\_para\_init

The description of pkcau\_hash\_struct\_para\_init is shown as below:

Table 3-574. Function pkcau\_hash\_struct\_para\_init

Function name	pkcau_hash_struct_para_init
Function prototype	void pkcau_hash_struct_para_init(pkcau_hash_parameter_struct* init_para);
Function descriptions	initialize hash parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	hash parameter struct, the structure members can refer to <a href="#">Table 3-563. Structure pkcau_hash_parameter_struct.</a>
Return value	
-	-

Example:

```
/* initialize PKCAU hash parameter struct with a default value */
```

```
pkcau_hash_parameter_struct pkcau_hash_parameter;
```

```
pkcau_hash_struct_para_init(&pkcau_hash_parameter);
```

### pkcau\_mod\_reduc\_struct\_para\_init

The description of pkcau\_mod\_reduc\_struct\_para\_init is shown as below:

Table 3-575. Function pkcau\_mod\_reduc\_struct\_para\_init

Function name	pkcau_mod_reduc_struct_para_init
Function prototype	void pkcau_mod_reduc_struct_para_init(pkcau_mod_reduc_parameter_struct* init_para);
Function descriptions	initialize modular reduction parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	modular reduction parameter struct, the structure members can refer to <a href="#">Table 3-557. Structure pkcau_mod_reduc_parameter_struct.</a>
Return value	
-	-

Example:

```
/* initialize PKCAU modular reduction parameter struct with a default value */
```

```
pkcau_mod_reduc_parameter_struct pkcau_mod_reduc_parameter;
```

```
pkcau_mod_reduc_struct_para_init(&pkcau_mod_reduc_parameter);
```

## pkcau\_enable

The description of pkcau\_enable is shown as below:

**Table 3-576. Function pkcau\_enable**

<b>Function name</b>	pkcau_enable
<b>Function prototype</b>	void pkcau_enable(void);
<b>Function descriptions</b>	enable PKCAU
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable PKCAU */
pkcau_enable();
```

## pkcau\_disable

The description of pkcau\_disable is shown as below:

**Table 3-577. Function pkcau\_disable**

<b>Function name</b>	pkcau_disable
<b>Function prototype</b>	void pkcau_disable(void);
<b>Function descriptions</b>	disable PKCAU
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable PKCAU */
```



pkcau\_disable();

## pkcau\_start

The description of pkcau\_start is shown as below:

**Table 3-578. Function pkcau\_start**

<b>Function name</b>	pkcau_start
<b>Function prototype</b>	void pkcau_start(void);
<b>Function descriptions</b>	start operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* start operation */
```

```
pkcau_start();
```

## pkcau\_mode\_set

The description of pkcau\_mode\_set is shown as below:

**Table 3-579. Function pkcau\_mode\_set**

<b>Function name</b>	pkcau_mode_set
<b>Function prototype</b>	void pkcau_mode_set(uint32_t mode);
<b>Function descriptions</b>	configure the PKCAU operation mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	PKCAU operation mode
<i>PKCAU_MODE_MOD_EXP</i>	Montgomery parameter computation then modular exponentiation
<i>PKCAU_MODE_MONT_PARAM</i>	Montgomery parameter computation only
<i>PKCAU_MODE_MOD_EXP_FAST</i>	modular exponentiation only
<i>PKCAU_MODE_CRT_EXP</i>	RSA CRT exponentiation
<i>PKCAU_MODE_MOD_</i>	modular inversion

<i>INVERSION</i>	
<i>PKCAU_MODE_ARITHMETIC_ADD</i>	arithmetic addition
<i>PKCAU_MODE_ARITHMETIC_SUB</i>	arithmetic subtraction
<i>PKCAU_MODE_ARITHMETIC_MUL</i>	arithmetic multiplication
<i>PKCAU_MODE_ARITHMETIC_COMP</i>	arithmetic comparison
<i>PKCAU_MODE_MODULAR_REDUCTION</i>	modular reduction
<i>PKCAU_MODE_MODULAR_ADD</i>	modular addition
<i>PKCAU_MODE_MODULAR_SUB</i>	modular subtraction
<i>PKCAU_MODE_MONTGOMERY_MUL</i>	Montgomery multiplication
<i>PKCAU_MODE_ECC_MUL</i>	Montgomery parameter computation then ECC scalar multiplication
<i>PKCAU_MODE_ECC_MUL_FAST</i>	ECC scalar multiplication only
<i>PKCAU_MODE_ECDSA_A_SIGN</i>	ECDSA sign
<i>PKCAU_MODE_ECDSA_A_VERIFICATION</i>	ECDSA verification
<i>PKCAU_MODE_POINT_CHECK</i>	point on elliptic curve Fp check
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PKCAU operation mode as PKCAU_MODE_ARITHMETIC_ADD */
pkcau_mode_set(PKCAU_MODE_ARITHMETIC_ADD);
```

## pkcau\_mont\_param\_operation

The description of pkcau\_mont\_param\_operation is shown as below:

**Table 3-580. Function pkcau\_mont\_param\_operation**

<b>Function name</b>	pkcau_mont_param_operation
<b>Function prototype</b>	void pkcau_mont_param_operation(pkcau_mont_parameter_struct

	*mont_para);
<b>Function descriptions</b>	execute montgomery parameter operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mont_para</b>	montgomery parameter struct, the structure members can refer to <a href="#">Table 3-553. Structure pkcau_mont_parameter_struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* initialize the montgomery parameter structure */

pkcau_mont_parameter_struct pkcau_mont_parameter;

pkcau_mont_struct_para_init(&pkcau_mont_parameter);

/* initialize the montgomery parameters */

pkcau_mont_parameter.modulus_len = ME_MOD_SIZE;

pkcau_mont_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute montgomery parameter operation */

pkcau_mont_param_operation(&pkcau_mont_parameter);

```

## pkcau\_mod\_operation

The description of pkcau\_mod\_operation is shown as below:

**Table 3-581. Function pkcau\_mod\_operation**

<b>Function name</b>	pkcau_mod_operation
<b>Function prototype</b>	void pkcau_mod_operation(pkcau_mod_parameter_struct *mod_para, uint32_t mode);
<b>Function descriptions</b>	execute modular operation, include modular addition, modular subtraction and montgomery multiplication
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mod_para</b>	modular parameter struct, the structure members can refer to <a href="#">Table 3-554. Structure pkcau_mod_parameter_struct.</a>
<b>Input parameter{in}</b>	
<b>mode</b>	modular operation mode
<b>PKCAU_MODE_MOD_</b>	modular addition

<i>ADD</i>	
<i>PKCAU_MODE_MOD_</i> <i>SUB</i>	modular subtraction
<i>PKCAU_MODE_MONT</i> <i>_MUL</i>	Montgomery multiplication
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* initialize the modular parameter structure */

pkcau_mod_parameter_struct pkcau_mod_parameter;

pkcau_mod_struct_para_init(&pkcau_mod_parameter);

/* initialize the modular parameters */

pkcau_mod_parameter.oprd_a = (uint8_t *)oprd_a;

pkcau_mod_parameter.oprd_b = (uint8_t *)oprd_b;

pkcau_mod_parameter.modulus_len = MA_MOD_SIZE;

pkcau_mod_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute modular addition operation */

pkcau_mod_operation(&pkcau_mod_parameter, PKCAU_MODE_MOD_ADD);

```

### pkcau\_mod\_exp\_operation

The description of pkcau\_mod\_exp\_operation is shown as below:

**Table 3-582. Function pkcau\_mod\_exp\_operation**

<b>Function name</b>	pkcau_mod_exp_operation
<b>Function prototype</b>	void pkcau_mod_exp_operation(pkcau_mod_exp_parameter_struct *mod_exp_para, uint32_t mode);
<b>Function descriptions</b>	execute modular exponentation operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mod_exp_para</b>	modular exponentation parameter struct, the structure members can refer to <a href="#">Table 3-555. Structure pkcau_mod_exp_parameter_struct</a> .
<b>Input parameter{in}</b>	
<b>mode</b>	modular exponentation operation mode
<i>PKCAU_MODE_MOD_</i>	montgomery parameter computation then modular exponentiation

<i>EXP</i>	
<i>PKCAU_MODE_MOD_</i> <i>EXP_FAST</i>	modular exponentiation only
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* initialize the modular exponentiation parameter structure */
pkcau_mod_exp_parameter_struct pkcau_mod_exp_parameter;
pkcau_mod_exp_struct_para_init(&pkcau_mod_exp_parameter);

/* initialize the modular exponentiation parameters */
pkcau_mod_exp_parameter.oprd_a = (uint8_t *)oprd_a;
pkcau_mod_exp_parameter.exp_e = (uint8_t *)exp_e;
pkcau_mod_exp_parameter.e_len = ME_E_SIZE;
pkcau_mod_exp_parameter.modulus_len = ME_MOD_SIZE;
pkcau_mod_exp_parameter.modulus_n = (uint8_t *)modulus_n;
pkcau_mod_exp_parameter.mont_para = (uint8_t *)mont_para;

/* execute modular exponentiation fast operation */
pkcau_mod_exp_operation(&pkcau_mod_exp_parameter,
PKCAU_MODE_MOD_EXP_FAST);

```

### pkcau\_mod\_inver\_operation

The description of pkcau\_mod\_inver\_operation is shown as below:

**Table 3-583. Function pkcau\_mod\_inver\_operation**

<b>Function name</b>	pkcau_mod_inver_operation
<b>Function prototype</b>	void void pkcau_mod_inver_operation(pkcau_mod_inver_parameter_struct *mod_inver_para);
<b>Function descriptions</b>	execute modular inversion operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mod_inver_para</b>	modular inversion parameter struct, the structure members can refer to <a href="#">Table 3-556. Structure pkcau_mod_inver_parameter_struct.</a>
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```

/* initialize the modular inversion parameter structure */

pkcau_mod_inver_parameter_struct pkcau_mod_inver_parameter;

pkcau_mod_inver_struct_para_init(&pkcau_mod_inver_parameter);

/* initialize the modular parameters */

pkcau_inver_exp_parameter.oprd_a = (uint8_t *)oprd_a;

pkcau_inver_exp_parameter.modulus_len = ME_MOD_SIZE;

pkcau_inver_exp_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute modular inversion operation */

pkcau_mod_inver_operation(&pkcau_mod_inver_parameter);

```

### pkcau\_mod\_reduc\_operation

The description of pkcau\_mod\_reduc\_operation is shown as below:

**Table 3-584. Function pkcau\_mod\_reduc\_operation**

<b>Function name</b>	pkcau_mod_reduc_operation
<b>Function prototype</b>	void pkcau_mod_reduc_operation(pkcau_mod_reduc_parameter_struct *mod_reduc_para);
<b>Function descriptions</b>	execute modular reduction operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mod_reduc_para</b>	modular reduction parameter struct, the structure members can refer to <a href="#">Table 3-557. Structure pkcau_mod_reduc_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* initialize the modular inversion parameter structure */

pkcau_mod_reduc_parameter_struct pkcau_mod_reduc_parameter;

pkcau_mod_reduc_struct_para_init(&pkcau_mod_reduc_parameter);

/* initialize the modular parameters */

```

```
pkcau_mod_reduc_parameter.oprd_a = (uint8_t *)oprd_a;

pkcau_mod_reduc_parameter.oprd_a_len = MA_A_SIZE;

pkcau_mod_reduc_parameter.modulus_len = MA_MOD_SIZE;

pkcau_mod_reduc_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute modular reduction operation */

pkcau_mod_reduc_operation(&pkcau_mod_reduc_parameter);
```

## pkcau\_arithmetic\_operation

The description of pkcau\_arithmetic\_operation is shown as below:

**Table 3-585. Function pkcau\_arithmetic\_operation**

<b>Function name</b>	pkcau_arithmetic_operation
<b>Function prototype</b>	void pkcau_arithmetic_operation(pkcau_arithmetic_parameter_struct *arithmetic_para, uint32_t mode);
<b>Function descriptions</b>	execute arithmetic operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>arithmetic_para</b>	arithmetic parameter struct, the structure members can refer to <a href="#">Table 3-558. Structure pkcau_arithmetic_parameter_struct</a> .
<b>Input parameter{in}</b>	
<b>mode</b>	arithmetic operation mode
PKCAU_MODE_ARITHMETIC_ADD	arithmetic addition
PKCAU_MODE_ARITHMETIC_SUB	arithmetic subtraction
PKCAU_MODE_ARITHMETIC_MUL	arithmetic multiplication
PKCAU_MODE_ARITHMETIC_COMP	arithmetic comparison
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the arithmetic parameter structure */

pkcau_arithmetic_parameter_struct pkcau_arithmetic_parameter;

pkcau_arithmetic_struct_para_init(&pkcau_arithmetic_parameter);
```

```
/* initialize the arithmetic addition parameters */

pkcau_ari_multi_parameter.oprd_a = (uint8_t *)oprd_a;

pkcau_ari_multi_parameter.oprd_b = (uint8_t *)oprd_b;

pkcau_ari_multi_parameter.oprd_len = AM_B_SIZE;

/* execute arithmetic addition operation */

pkcau_arithmetic_operation(&pkcau_arithmetic_parameter,
PKCAU_MODE_ARITHMETIC_ADD);
```

## pkcau\_crt\_exp\_operation

The description of pkcau\_crt\_exp\_operation is shown as below:

**Table 3-586. Function pkcau\_crt\_exp\_operation**

<b>Function name</b>	pkcau_crt_exp_operation
<b>Function prototype</b>	void pkcau_crt_exp_operation(pkcau_crt_parameter_struct* crt_para);
<b>Function descriptions</b>	execute RSA CRT exponentiation operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>crt_para</b>	CRT parameter struct, the structure members can refer to <a href="#">Table 3-559. Structure pkcau_crt_parameter_struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the arithmetic parameter structure */

pkcau_crt_parameter_struct pkcau_crt_parameter;

pkcau_crt_exp_operation(&pkcau_crt_parameter);

/* initialize the input ECC curve parameters */

pkcau_crt_parameter.oprd_a = (uint8_t *)rsa_crt_a;

pkcau_crt_parameter.oprd_dp = (uint8_t *)rsa_crt_dp;

pkcau_crt_parameter.oprd_dq = (uint8_t *)rsa_crt_dq;

pkcau_crt_parameter.oprd_qinv = (uint8_t *)rsa_crt_qinv;

pkcau_crt_parameter.oprd_p = (uint8_t *)rsa_crt_p;

pkcau_crt_parameter.oprd_q = (uint8_t *)rsa_crt_q;
```



```
pkcau crt_parameter.oprd_len = sizeof(rsa crt_a);
```

```
/* execute RSA CRT exponentiation operation */
```

```
pkcau crt_exp_operation(&pkcau crt_parameter);
```

## pkcau\_point\_check\_operation

The description of pkcau\_point\_check\_operation is shown as below:

**Table 3-587. Function pkcau\_point\_check\_operation**

<b>Function name</b>	pkcau_point_check_operation
<b>Function prototype</b>	void pkcau_point_check_operation(pkcau_point_parameter_struct* point_para, const pkcau_ec_group_parameter_struct* curve_group_para);
<b>Function descriptions</b>	execute point check operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>point_para</b>	point parameter struct, the structure members can refer to <a href="#">Table 3-561. Structure pkcau_point_parameter_struct</a> .
<b>Input parameter{in}</b>	
<b>curve_group_para</b>	ECC curve parameter struct, the structure members can refer to <a href="#">Table 3-560. Structure pkcau_ec_group_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize point parameter and ECC curve parameter structure */
```

```
pkcau_point_parameter_struct pkcau_point_parameter;
```

```
pkcau_ec_group_parameter_struct pkcau_curve_group;
```

```
pkcau_point_struct_para_init(&pkcau_point_parameter);
```

```
pkcau_ec_group_struct_para_init(&pkcau_curve_group);
```

```
/* initialize the input point parameter */
```

```
pkcau_point_check_parameter.point_x = pcheck_x;
```

```
pkcau_point_check_parameter.point_y = pcheck_y;
```

```
/* initialize the input ECC curve parameter */
```

```
pkcau_curve_group.modulus_p = (uint8_t *)brainpoolp256r1_p;
```

```
pkcau_curve_group.coff_a = (uint8_t *)brainpoolp256r1_a;
```

```
pkcau_curve_group.coff_b = (uint8_t *)brainpoolp256r1_b;

pkcau_curve_group.a_sign = 0;

pkcau_curve_group.modulus_p_len = sizeof(brainpoolp256r1_p);

/* execute point check operation */

pkcau_point_check_operation(&pkcau_point_parameter, &pkcau_curve_group);
```

## pkcau\_point\_mul\_operation

The description of pkcau\_point\_mul\_operation is shown as below:

**Table 3-588. Function pkcau\_point\_mul\_operation**

Function name	pkcau_point_mul_operation
Function prototype	void pkcau_point_mul_operation(pkcau_point_parameter_struct *point_para, const pkcau_ec_group_parameter_struct* curve_group_para, uint32_t mode);
Function descriptions	execute point multiplication operation
Precondition	-
The called functions	-
Input parameter{in}	
point_para	point parameter struct, the structure members can refer to <a href="#">Table 3-561. Structure pkcau_point_parameter_struct</a> .
Input parameter{in}	
curve_group_para	ECC curve parameter struct, the structure members can refer to <a href="#">Table 3-560. Structure pkcau_ec_group_parameter_struct</a> .
Input parameter{in}	
mode	point multiplication operation mode
PKCAU_MODE_ECC_MUL	montgomery parameter computation then ECC scalar multiplication
PKCAU_MODE_ECC_MUL_FAST	ECC scalar multiplication only
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize point parameter and ECC curve parameter structure */

pkcau_point_parameter_struct pkcau_point_parameter;

pkcau_ec_group_parameter_struct pkcau_curve_group;

pkcau_point_struct_para_init(&pkcau_point_parameter);
```

```
pkcau_ec_group_struct_para_init(&pkcau_curve_group);

/* initialize the input point parameter */

pkcau_point_parameter.point_x = ec_pmul_x;

pkcau_point_parameter.point_y = ec_pmul_y;

/* initialize the input ECC curve parameter */

pkcau_curve_group.modulus_p = (uint8_t *)brainpoolp256r1_p;

pkcau_curve_group.coff_a = (uint8_t *)brainpoolp256r1_a;

pkcau_curve_group.modulus_p_len = sizeof(brainpoolp256r1_p);

pkcau_curve_group.a_sign = 0;

pkcau_curve_group.multi_k = ec_pmul_k;

pkcau_curve_group.k_len = PMUL_K_SIZE;

/* execute scalar multiplication operation */

pkcau_point_mul_operation(&pkcau_point_parameter, &pkcau_curve_group,
PKCAU_MODE_ECC_MUL);
```

### pkcau\_ecdsa\_sign\_operation

The description of pkcau\_ecdsa\_sign\_operation is shown as below:

**Table 3-589. Function pkcau\_ecdsa\_sign\_operation**

Function name	pkcau_ecdsa_sign_operation
Function prototype	void pkcau_ecdsa_sign_operation(const uint8_t* p_key_d, const uint8_t* k, pkcau_hash_parameter_struct* hash_para, const pkcau_ec_group_parameter_struct* curve_group_para);
Function descriptions	execute ECDSA sign operation
Precondition	-
The called functions	-
Input parameter{in}	
p_key_d	private key d
Input parameter{in}	
k	integer k
Input parameter{in}	
hash_para	hash parameter struct, the structure members can refer to <a href="#">Table 3-563. Structure pkcau_hash_parameter_struct</a> .
Input parameter{in}	
curve_group_para	ECC curve parameter struct, the structure members can refer to <a href="#">Table 3-560. Structure pkcau_ec_group_parameter_struct</a> .
Output parameter{out}	

-	-
Return value	
-	-

Example:

```

/* initialize hash parameter and ECC curve parameter structure */

pkcau_hash_parameter_struct pkcau_hash_parameter;

pkcau_ec_group_parameter_struct pkcau_curve_group;

pkcau_hash_struct_para_init(&pkcau_hash_parameter);

pkcau_ec_group_struct_para_init(&pkcau_curve_group);

/* initialize the input hash parameter */

pkcau_hash_parameter.hash_z = hash;

pkcau_hash_parameter.hash_z_len = DATA_SIZE;

/* initialize the input ECC curve parameter */

pkcau_curve_group.modulus_p = secp112r2_p;

pkcau_curve_group.coff_a = secp112r2_a;

pkcau_curve_group.a_sign = 0;

pkcau_curve_group.base_point_x = secp112r2_gx;

pkcau_curve_group.base_point_y = secp112r2_gy;

pkcau_curve_group.order_n = secp112r2_n,

pkcau_curve_group.modulus_p_len = sizeof(secp112r2_p);

pkcau_curve_group.order_n_len = sizeof(secp112r2_n);

/* execute ECDSA sign operation */

pkcau_ecdsa_sign_operation(d, k, &pkcau_hash_parameter, &pkcau_curve_group);

```

### pkcau\_ecdsa\_verification\_operation

The description of pkcau\_ecdsa\_verification\_operation is shown as below:

**Table 3-590. Function pkcau\_ecdsa\_verification\_operation**

Function name	pkcau_ecdsa_verification_operation
Function prototype	void pkcau_ecdsa_verification_operation(pkcau_point_parameter_struct *point_para, pkcau_hash_parameter_struct *hash_para, pkcau_signature_parameter_struct *signature_para, const pkcau_ec_group_parameter_struct* curve_group_para);

<b>Function descriptions</b>	execute ECDSA verification operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>point_para</b>	point parameter struct, the structure members can refer to <a href="#">Table 3-561. Structure pkcau_point_parameter_struct.</a>
<b>Input parameter{in}</b>	
<b>hash_para</b>	hash parameter struct, the structure members can refer to <a href="#">Table 3-563. Structure pkcau_hash_parameter_struct.</a>
<b>Input parameter{in}</b>	
<b>signature_para</b>	signature parameter struct, the structure members can refer to <a href="#">Table 3-562. Structure pkcau_signature_parameter_struct.</a>
<b>Input parameter{in}</b>	
<b>curve_group_para</b>	ECC curve parameter struct, the structure members can refer to <a href="#">Table 3-560. Structure pkcau_ec_group_parameter_struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* initialize point parameter, hash parameter and ECC curve parameter structure */
pkcau_point_parameter_struct pkcau_point_parameter;
pkcau_hash_parameter_struct pkcau_hash_parameter;
pkcau_signature_parameter_struct pkcau_signature_parameter;
pkcau_ec_group_parameter_struct pkcau_curve_group;
pkcau_point_struct_para_init(&pkcau_point_parameter);
pkcau_hash_struct_para_init(&pkcau_hash_parameter);
pkcau_signature_struct_para_init(&pkcau_signature_parameter);
pkcau_ec_group_struct_para_init(&pkcau_curve_group);

/* initialize the input point parameters */
pkcau_point_parameter.point_x = ecc_verify_x;
pkcau_point_parameter.point_y = ecc_verify_y;

/* initialize the input hash parameters */
pkcau_hash_parameter.hash_z = (uint8_t *)ecc_verify_hash;
pkcau_hash_parameter.hash_z_len = sizeof(ecc_verify_hash);

```

```

/* initialize the input ECC signature parameters */

pkcau_signature_parameter.sign_r = (uint8_t *)ecc_verify_r;

pkcau_signature_parameter.sign_s = (uint8_t *)ecc_verify_s;

/* initialize the input ECC curve parameters */

pkcau_curve_group.modulus_p = (uint8_t *)brainpoolp256r1_p;

pkcau_curve_group.coff_a = (uint8_t *)brainpoolp256r1_a;

pkcau_curve_group.a_sign = 0;

pkcau_curve_group.base_point_x = (uint8_t *)brainpoolp256r1_gx;

pkcau_curve_group.base_point_y = (uint8_t *)brainpoolp256r1_gy;

pkcau_curve_group.order_n = (uint8_t *)brainpoolp256r1_n,

pkcau_curve_group.modulus_p_len = sizeof(brainpoolp256r1_p);

pkcau_curve_group.order_n_len = sizeof(brainpoolp256r1_n);

/* execute ECDSA verification operation */

pkcau_ecdsa_verification_operation(&pkcau_point_parameter, &pkcau_hash_parameter,
&pkcau_signature_parameter, &pkcau_curve_group);

```

## pkcau\_memread

The description of pkcau\_memread is shown as below:

**Table 3-591. Function pkcau\_memread**

Function name	pkcau_memread
Function prototype	void pkcau_memread(uint32_t offset, uint8_t buf[], uint32_t size);
Function descriptions	read result from PKCAU RAM
Precondition	-
The called functions	-
Input parameter{in}	
offset	RAM address offset to PKCAU base address
Input parameter{in}	
size	number of byte to read
Output parameter{out}	
buf	big endian result buffer, the left most byte from the PKCAU should be in the first element of buffer
Return value	
-	-

Example:

```

/* read results from RAM address */

```

```
uint8_t verify_res = 0;
```

```
pkcau_memread(0x5B0, &verify_res, 1);
```

## pkcau\_flag\_get

The description of pkcau\_flag\_get is shown as below:

**Table 3-592. Function pkcau\_flag\_get**

<b>Function name</b>	pkcau_flag_get
<b>Function prototype</b>	FlagStatus pkcau_flag_get(uint32_t flag);
<b>Function descriptions</b>	get PKCAU flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	PKCAU flags
<i>PKCAU_FLAG_ADDRE RR</i>	address error flag
<i>PKCAU_FLAG_RAME RR</i>	PKCAU RAM error flag
<i>PKCAU_FLAG_END</i>	end of PKCAU operation flag
<i>PKCAU_FLAG_BUSY</i>	busy flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get PKCAU flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = pkcau_flag_get(PKCAU_FLAG_ADDRERR);
```

## pkcau\_flag\_clear

The description of pkcau\_flag\_clear is shown as below:

**Table 3-593. Function pkcau\_flag\_clear**

<b>Function name</b>	pkcau_flag_clear
<b>Function prototype</b>	void pkcau_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear PKCAU flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	PKCAU flags

<i>PKCAU_FLAG_ADDRE RR</i>	address error flag
<i>PKCAU_FLAG_RAME RR</i>	PKCAU RAM error flag
<i>PKCAU_FLAG_END</i>	end of PKCAU operation flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear address error flag*/
```

```
pkcau_flag_clear(PKCAU_FLAG_ADDRERR);
```

### pkcau\_interrupt\_enable

The description of pkcau\_interrupt\_enable is shown as below:

**Table 3-594. Function pkcau\_interrupt\_enable**

<b>Function name</b>	pkcau_interrupt_enable
<b>Function prototype</b>	void pkcau_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable PKCAU interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type
<i>PKCAU_INT_ADDRER R</i>	address error interrupt
<i>PKCAU_INT_RAMERR</i>	PKCAU RAM error interrupt
<i>PKCAU_INT_END</i>	end of PKCAU operation interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable PKCAU address error interrupt */
```

```
pkcau_interrupt_enable(PKCAU_INT_ADDRERR);
```

### pkcau\_interrupt\_disable

The description of pkcau\_interrupt\_disable is shown as below:



Table 3-595. Function pkcau\_interrupt\_disable

<b>Function name</b>	pkcau_interrupt_disable
<b>Function prototype</b>	void pkcau_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable PKCAU interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type
<i>PKCAU_INT_ADDRER R</i>	address error interrupt
<i>PKCAU_INT_RAMERR</i>	PKCAU RAM error interrupt
<i>PKCAU_INT_END</i>	end of PKCAU operation interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable PKCAU address error interrupt */
pkcau_interrupt_disable(PKCAU_INT_ADDRERR);
```

### pkcau\_interrupt\_flag\_get

The description of pkcau\_interrupt\_flag\_get is shown as below:

Table 3-596. Function pkcau\_interrupt\_flag\_get

<b>Function name</b>	pkcau_interrupt_flag_get
<b>Function prototype</b>	FlagStatus pkcau_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	get PKCAU interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	PKCAU interrupt flags
<i>PKCAU_INT_FLAG_AD DRERR</i>	address error interrupt flag
<i>PKCAU_INT_FLAG_RA MERR</i>	PKCAU RAM error interrupt flag
<i>PKCAU_INT_FLAG_EN D</i>	end of PKCAU operation interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get PKCAU interrupt flag status */

FlagStatus flag_state = RESET;

flag_state = pkcau_interrupt_flag_get(PKCAU_INT_FLAG_ADDRERR);
```

## pkcau\_interrupt\_flag\_clear

The description of pkcau\_interrupt\_flag\_clear is shown as below:

**Table 3-597. Function pkcau\_interrupt\_flag\_clear**

<b>Function name</b>	pkcau_interrupt_flag_clear
<b>Function prototype</b>	void pkcau_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear PKCAU flag interrupt status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	PKCAU interrupt flags
<i>PKCAU_INT_FLAG_ADDRERR</i>	address error interrupt flag
<i>PKCAU_INT_FLAG_RAMERR</i>	PKCAU RAM error interrupt flag
<i>PKCAU_INT_FLAG_END</i>	end of PKCAU operation interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear address error interrupt flag*/

pkcau_interrupt_flag_clear(PKCAU_INT_FLAG_ADDRERR);
```

## 3.20. PMU

According to the Power management unit (PMU), provides five types of power saving modes, including Sleep, Deep-sleep, SRAM\_sleep, WIFI\_sleep and Standby mode. The PMU registers are listed in chapter [3.20.1](#), the PMU firmware functions are introduced in chapter [3.20.2](#).

### 3.20.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

Table 3-598. 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
PMU_RFCTL	PMU RF control register
PMU_SECCFG	PMU secure configuration register
PMU_PRIVCFG	PMU privilege configuration register
PMU_DSLVC_ULK KEY	Deepsleep voltage configure unlock register

### 3.20.2. Descriptions of Peripheral functions

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

Table 3-599. PMU firmware function

Function name	Function description
pmu_deinit	deinitialize the PMU
pmu_lvd_select	select low voltage detector threshold
pmu_deepsleep_voltage_select	select deep-sleep mode core voltage
pmu_lvd_disable	disable PMU lvd
pmu_vlvd_enable	enable PMU VLVD
pmu_vlvd_disable	disable PMU VLVD
pmu_ldo_output_select	select LDO output voltage
pmu_to_sleepmode	PMU work at sleep mode
pmu_to_deepsleepmode	PMU work at deepsleep mode
pmu_to_standbymode	pmu work at standby mode
pmu_wakeup_pin_enable	enable wakeup pin
pmu_wakeup_pin_disable	disable wakeup pin
pmu_wakeup_pin_edge_select	wakeup pin edge selection
pmu_backup_write_enable	enable backup domain write
pmu_backup_write_disable	disable backup domain write
pmu_wifi_power_enable	enable WIFI power
pmu_wifi_power_disable	disable WIFI power
pmu_wifi_low_power_control	WIFI low power control
pmu_sram_low_power_control	SRAM low power control
pmu_rf_force_enable	RF sequence force open/close
pmu_rf_force_disable	disable RF sequence open/close force
pmu_rf_sequence_config	RF sequence configuration
pmu_security_enable	enable the security attribution
pmu_security_disable	disable the security attribution
pmu_privilege_enable	enable the privileged access

Function name	Function description
pmu_privilege_disable	disable the privileged access
pmu_flag_reset	reset flag bit
pmu_flag_get	get flag state

## pmu\_deinit

The description of pmu\_deinit is shown as below:

**Table 3-600. 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 3-601. Function pmu\_lvd\_select**

Function name	pmu_lvd_select
Function prototype	void pmu_lvd_select(uint32_t lvd_t_n);
Function descriptions	select low voltage detector threshold
Precondition	-
The called functions	-
Input parameter{in}	
lvd_t_n	voltage threshold value
PMU_LVDT_0	voltage threshold is 2.1V
PMU_LVDT_1	voltage threshold is 2.3V
PMU_LVDT_2	voltage threshold is 2.4V
PMU_LVDT_3	voltage threshold is 2.6V
PMU_LVDT_4	voltage threshold is 2.7V
PMU_LVDT_5	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select low voltage detector threshold as 3.1V */
```

```
pmu_lvd_select (PMU_LVDT_7);
```

### pmu\_deepsleep\_voltage\_select

The description of pmu\_deepsleep\_voltage\_select is shown as below:

**Table 3-602. Function pmu\_deepsleep\_voltage\_select**

<b>Function name</b>	pmu_deepsleep_voltage_select
<b>Function prototype</b>	void pmu_deepsleep_voltage_select(uint32_t voltage_n);
<b>Function descriptions</b>	select deep-sleep mode core voltage
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>voltage_n</b>	core voltage in Deep-sleep mode
<i>PMU_DSLPVS_1_1</i>	core voltage is 1.1V in Deep-sleep mode
<i>PMU_DSLPVS_1_0</i>	core voltage is 1.0V in Deep-sleep mode
<i>PMU_DSLPVS_0_9</i>	core voltage is 0.9V in Deep-sleep mode
<i>PMU_DSLPVS_0_8</i>	core voltage is 0.8V in Deep-sleep mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select core voltage in Deep-sleep mode as 1.1V */
```

```
pmu_deepsleep_voltage_select (PMU_DSLPVS_1_1);
```

### pmu\_lvd\_disable

The description of pmu\_lvd\_disable is shown as below:

**Table 3-603. Function pmu\_lvd\_disable**

<b>Function name</b>	pmu_lvd_disable
<b>Function prototype</b>	void pmu_lvd_disable(void);
<b>Function descriptions</b>	disable PMU lvd

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable PMU lvd */
```

```
pmu_lvd_disable();
```

### pmu\_vlvd\_enable

The description of pmu\_vlvd\_enable is shown as below:

**Table 3-604. Function pmu\_vlvd\_enable**

<b>Function name</b>	pmu_vlvd_enable
<b>Function prototype</b>	void pmu_vlvd_enable(void);
<b>Function descriptions</b>	enable PMU VLVD
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable PMU VLVD */
```

```
pmu_vlvd_enable();
```

### pmu\_vlvd\_disable

The description of pmu\_vlvd\_disable is shown as below:

**Table 3-605. Function pmu\_vlvd\_disable**

<b>Function name</b>	pmu_vlvd_disable
<b>Function prototype</b>	void pmu_vlvd_disable(void);
<b>Function descriptions</b>	disable PMU VLVD
<b>Precondition</b>	-
<b>The called functions</b>	-

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

Example:

```
/* disable PMU VLVD */
```

```
pmu_vlvd_disable();
```

### pmu\_ldo\_output\_select

The description of pmu\_ldo\_output\_select is shown as below:

**Table 3-606. Function pmu\_ldo\_output\_select**

Function name	pmu_ldo_output_select
Function prototype	void pmu_ldo_output_select(uint32_t ldo_output);
Function descriptions	select LDO output voltage
Precondition	-
The called functions	-
Input parameter{in}	
ldo_output	LDO output voltage mode
PMU_LDOVS_LOW	LDO output voltage low mode
PMU_LDOVS_HIGH	LDO output voltage high mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* LDO output low voltage */
```

```
pmu_ldo_output_select(PMU_LDOVS_LOW);
```

### pmu\_to\_sleepmode

The description of pmu\_to\_sleepmode is shown as below:

**Table 3-607. Function pmu\_to\_sleepmode**

Function name	pmu_to_sleepmode
Function prototype	void pmu_to_sleepmode(uint8_t sleepmodecmd);
Function descriptions	PMU work at sleep mode
Precondition	-
The called functions	-

Input parameter{in}	
<b>sleepmodecmd</b>	command to enter sleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PMU work at sleep mode */
pmu_to_sleepmode(WFI_CMD);
```

### pmu\_to\_deepsleepmode

The description of pmu\_to\_deepsleepmode is shown as below:

**Table 3-608. Function pmu\_to\_deepsleepmode**

<b>Function name</b>	pmu_to_deepsleepmode
<b>Function prototype</b>	void pmu_to_deepsleepmode(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmodecmd);
<b>Function descriptions</b>	PMU work at deepsleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>ldo</b>	ldo work mode
<i>PMU_LDO_NORMAL</i>	LDO normal work when pmu enter deepsleep mode
<i>PMU_LDO_LOWPOWER</i>	LDO work at low power mode when pmu enter deepsleep mode
Input parameter{in}	
<b>lowdrive</b>	ldo low-driver mode
<i>PMU_LOWDRIIVER_DISABLE</i>	Low-driver mode disable in deep-sleep mode
<i>PMU_LOWDRIIVER_ENABLE</i>	Low-driver mode enable in deep-sleep mode
Input parameter{in}	
<b>deepsleepmodecmd</b>	command to enter deepsleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
Output parameter{out}	
-	-
Return value	
-	-



Example:

```
/* PMU work at deepsleep mode */
```

```
pmu_to_deepsleepmode(PMU_LDO_NORMAL, PMU_LOWDRIVER_ENABLE, WFI_CMD);
```

## pmu\_to\_standbymode

The description of pmu\_to\_standbymode is shown as below:

**Table 3-609. Function pmu\_to\_standbymode**

<b>Function name</b>	pmu_to_standbymode
<b>Function prototype</b>	void pmu_to_standbymode(void);
<b>Function descriptions</b>	pmu work at standby mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work at standby mode */
```

```
pmu_to_standbymode();
```

## pmu\_wakeup\_pin\_enable

The description of pmu\_wakeup\_pin\_enable is shown as below:

**Table 3-610. Function pmu\_wakeup\_pin\_enable**

<b>Function name</b>	pmu_wakeup_pin_enable
<b>Function prototype</b>	void pmu_wakeup_pin_enable(uint32_t wakeup_pin);
<b>Function descriptions</b>	enable wakeup pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_pin</b>	the pin to wakeup system
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA2)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PA15)
PMU_WAKEUP_PIN2	WKUP Pin 2 (PB2)
PMU_WAKEUP_PIN3	WKUP Pin 3 (PA12)
PMU_WAKEUP_PIN4	WKUP Pin 0 (PD5)
PMU_WAKEUP_PIN5	WKUP Pin 1 (PD4)

<i>PMU_WAKEUP_PIN6</i>	WKUP Pin 2 (PB0)
<i>PMU_WAKEUP_PIN7</i>	WKUP Pin 3 (PD3)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-611. Function pmu\_wakeup\_pin\_disable**

<b>Function name</b>	pmu_wakeup_pin_disable
<b>Function prototype</b>	void pmu_wakeup_pin_disable(uint32_t wakeup_pin);
<b>Function descriptions</b>	disable wakeup pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_pin</b>	the pin to wakeup system
<i>PMU_WAKEUP_PIN0</i>	WKUP Pin 0 (PA2)
<i>PMU_WAKEUP_PIN1</i>	WKUP Pin 1 (PA15)
<i>PMU_WAKEUP_PIN2</i>	WKUP Pin 2 (PB2)
<i>PMU_WAKEUP_PIN3</i>	WKUP Pin 3 (PA12)
<i>PMU_WAKEUP_PIN4</i>	WKUP Pin 0 (PD5)
<i>PMU_WAKEUP_PIN5</i>	WKUP Pin 1 (PD4)
<i>PMU_WAKEUP_PIN6</i>	WKUP Pin 2 (PB0)
<i>PMU_WAKEUP_PIN7</i>	WKUP Pin 3 (PD3)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable wakeup pin 0 */
```

```
pmu_wakeup_pin_disable(PMU_WAKEUP_PIN0);
```

## pmu\_wakeup\_pin\_edge\_select

The description of pmu\_wakeup\_pin\_edge\_select is shown as below:

Table 3-612. Function pmu\_wakeup\_pin\_edge\_select

<b>Function name</b>	pmu_wakeup_pin_edge_select
<b>Function prototype</b>	void pmu_wakeup_pin_edge_select(uint32_t wakeup_pin_edge, uint32_t edge_type);
<b>Function descriptions</b>	wakeup pin edge selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_pin_edge</b>	wakeup pin edge selection
<i>PMU_WAKEUP_PIN0_EDGE</i>	WKUP Pin 0 (PA2) edge selection
<i>PMU_WAKEUP_PIN1_EDGE</i>	WKUP Pin 1 (PA15) edge selection
<i>PMU_WAKEUP_PIN2_EDGE</i>	WKUP Pin 2 (PB2) edge selection
<i>PMU_WAKEUP_PIN3_EDGE</i>	WKUP Pin 3 (PA12) edge selection
<i>PMU_WAKEUP_PIN4_EDGE</i>	WKUP Pin 4 (PD5) edge selection
<i>PMU_WAKEUP_PIN5_EDGE</i>	WKUP Pin 5 (PD4) edge selection
<i>PMU_WAKEUP_PIN6_EDGE</i>	WKUP Pin 6 (PB0) edge selection
<i>PMU_WAKEUP_PIN7_EDGE</i>	WKUP Pin 7 (PD3) edge selection
<b>Input parameter{in}</b>	
<b>edge_type</b>	wakeup pin edge type
<i>WAKEUP_PIN_EDGE_RISING</i>	WKUP pin active on rising edge
<i>WAKEUP_PIN_EDGE_FALLING</i>	WKUP pin active on falling edge
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select wakeup pin 0 edge type as rising*/
```

```
pmu_wakeup_pin_edge_select(PMU_WAKEUP_PIN0_EDGE,
WAKEUP_PIN_EDGE_RISING);
```

## pmu\_backup\_write\_enable

The description of pmu\_backup\_write\_enable is shown as below:

**Table 3-613. Function pmu\_backup\_write\_enable**

<b>Function name</b>	pmu_backup_write_enable
<b>Function prototype</b>	void pmu_backup_write_enable(void);
<b>Function descriptions</b>	enable backup domain write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-614. Function pmu\_backup\_write\_disable**

<b>Function name</b>	pmu_backup_write_disable
<b>Function prototype</b>	void pmu_backup_write_disable(void);
<b>Function descriptions</b>	disable backup domain write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable backup domain write */
pmu_backup_write_disable();
```

## pmu\_wifi\_power\_enable

The description of pmu\_wifi\_power\_enable is shown as below:

**Table 3-615. Function pmu\_wifi\_power\_enable**

<b>Function name</b>	pmu_wifi_power_enable
<b>Function prototype</b>	void pmu_wifi_power_enable(void);
<b>Function descriptions</b>	WIFI power enable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable WIFI power */
pmu_wifi_power_enable();
```

## pmu\_wifi\_power\_disable

The description of pmu\_wifi\_power\_disable is shown as below:

**Table 3-616. Function pmu\_wifi\_power\_disable**

<b>Function name</b>	pmu_wifi_power_disable
<b>Function prototype</b>	void pmu_wifi_power_disable(void);
<b>Function descriptions</b>	WIFI power disable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable WIFI power */
pmu_wifi_power_disable();
```

## pmu\_wifi\_low\_power\_control

The description of pmu\_wifi\_low\_power\_control is shown as below:

**Table 3-617. Function pmu\_wifi\_low\_power\_control**

<b>Function name</b>	pmu_wifi_low_power_control
<b>Function prototype</b>	void pmu_wifi_low_power_control(uint32_t wifi_lpwr_cfg);
<b>Function descriptions</b>	WIFI low power control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
wifi_lpwr_cfg	WIFI sleep control
PMU_WIFI_SLEEP	WIFI go to sleep
PMU_WIFI_WAKE	WIFI wakeup
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* WIFI go to sleep */
```

```
pmu_wifi_low_power_control(PMU_WIFI_SLEEP);
```

## pmu\_sram\_low\_power\_control

The description of pmu\_sram\_low\_power\_control is shown as below:

**Table 3-618. Function pmu\_sram\_low\_power\_control**

<b>Function name</b>	pmu_sram_low_power_control
<b>Function prototype</b>	void pmu_sram_low_power_control(uint32_t sram_lpwr_cfg);
<b>Function descriptions</b>	SRAM low power control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
sram_lpwr_cfg	SRAM sleep control
PMU_SRAM1_SLEEP	SRAM1 go to sleep
PMU_SRAM1_WAKE	SRAM1 wakeup
PMU_SRAM2_SLEEP	SRAM2 go to sleep
PMU_SRAM2_WAKE	SRAM2 wakeup
PMU_SRAM3_SLEEP	SRAM3 go to sleep
PMU_SRAM3_WAKE	SRAM3 wakeup
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* SRAM2 go to sleep */
```

```
pmu_sram_low_power_control(PMU_SRAM2_SLEEP);
```

### pmu\_rf\_force\_enable

The description of pmu\_rf\_force\_enable is shown as below:

**Table 3-619. Function pmu\_rf\_force\_enable**

<b>Function name</b>	pmu_rf_force_enable
<b>Function prototype</b>	void pmu_rf_force_enable(uint32_t force);
<b>Function descriptions</b>	enable RF sequence force open/close
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>force</b>	enable RF power control
<i>PMU_RF_FORCE_OPEN</i>	Software force start, open RF power
<i>PMU_RF_FORCE_CLOSE</i>	Software force close, close RF power
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable open RF power */
```

```
pmu_rf_force_enable(PMU_RF_FORCE_OPEN);
```

### pmu\_rf\_force\_disable

The description of pmu\_rf\_force\_disable is shown as below:

**Table 3-620. Function pmu\_rf\_force\_disable**

<b>Function name</b>	pmu_rf_force_disable
<b>Function prototype</b>	void pmu_rf_force_disable(uint32_t force);
<b>Function descriptions</b>	disable RF sequence open/close force
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>force</b>	disable RF power control
<i>PMU_RF_FORCE_OPEN</i>	Software force start, open RF power

<i>EN</i>	
<i>PMU_RF_FORCE_CLOSE</i>	Software force close, close RF power
<i>OSE</i>	
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable open RF power */
```

```
pmu_rf_force_disable(PMU_RF_FORCE_OPEN);
```

## pmu\_rf\_sequence\_config

The description of pmu\_rf\_sequence\_config is shown as below:

**Table 3-621. Function pmu\_rf\_sequence\_config**

<b>Function name</b>	pmu_rf_sequence_config
<b>Function prototype</b>	void pmu_rf_sequence_config(uint32_t mode);
<b>Function descriptions</b>	RF sequence configuration
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	RF sequence mode
<i>PMU_RF_SOFTWARE</i>	RF software sequence enable
<i>PMU_RF_HARDWARE</i>	RF hardware sequence enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable RF hardware sequence */
```

```
pmu_rf_sequence_config(PMU_RF_HARDWARE);
```

## pmu\_security\_enable

The description of pmu\_security\_enable is shown as below:

**Table 3-622. Function pmu\_security\_enable**

<b>Function name</b>	pmu_security_enable
<b>Function prototype</b>	void pmu_security_enable(uint32_t security);
<b>Function descriptions</b>	enable the security attribution
<b>Precondition</b>	-



<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>security</b>	security attribution configuration
<i>PMU_SEC_WAKEUP_PIN0</i>	WKUP pin 0 security
<i>PMU_SEC_WAKEUP_PIN1</i>	WKUP pin 1 security
<i>PMU_SEC_WAKEUP_PIN2</i>	WKUP pin 2 security
<i>PMU_SEC_WAKEUP_PIN3</i>	WKUP pin 3 security
<i>PMU_SEC_LPLDO_DP_SLP_STB</i>	Low-power mode security
<i>PMU_SEC_LVD_VLVD</i>	Voltage detection and monitoring security
<i>PMU_SEC_BACKUP_WRITE</i>	Backup Domain write access security
<i>PMU_SEC_WIFI_SRAM_CONTROL</i>	WiFi_sleep and SRAM_sleep mode security
<i>PMU_SEC_RF_SEQUENCE</i>	RF security
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable WKUP pin 0 security */
```

```
pmu_security_enable(PMU_SEC_WAKEUP_PIN0);
```

### pmu\_security\_disable

The description of pmu\_security\_disable is shown as below:

**Table 3-623. Function pmu\_security\_disable**

<b>Function name</b>	pmu_security_disable
<b>Function prototype</b>	void pmu_security_disable(uint32_t security);
<b>Function descriptions</b>	disable the security attribution
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>security</b>	security attribution configuration
<i>PMU_SEC_WAKEUP_PIN0</i>	WKUP pin 0 security

<i>PMU_SEC_WAKEUP_PIN1</i>	WKUP pin 1 security
<i>PMU_SEC_WAKEUP_PIN2</i>	WKUP pin 2 security
<i>PMU_SEC_WAKEUP_PIN3</i>	WKUP pin 3 security
<i>PMU_SEC_LPLDO_DP_SLP_STB</i>	Low-power mode security
<i>PMU_SEC_LVD_VLVD</i>	Voltage detection and monitoring security
<i>PMU_SEC_BACKUP_WRITE</i>	Backup Domain write access security
<i>PMU_SEC_WIFI_SRAM_CONTROL</i>	WIFI_sleep and SRAM_sleep mode security
<i>PMU_SEC_RF_SEQUENCE</i>	RF security
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable WKUP pin 0 security */
```

```
pmu_security_disable(PMU_SEC_WAKEUP_PIN0);
```

## pmu\_privilege\_enable

The description of pmu\_privilege\_enable is shown as below:

**Table 3-624. Function pmu\_privilege\_enable**

<b>Function name</b>	pmu_privilege_enable
<b>Function prototype</b>	void pmu_privilege_enable(void);
<b>Function descriptions</b>	enable the privileged access only
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the privileged access only */
```

pmu\_privilege\_enable();

## pmu\_privilege\_disable

The description of pmu\_privilege\_disable is shown as below:

**Table 3-625. Function pmu\_privilege\_disable**

<b>Function name</b>	pmu_privilege_disable
<b>Function prototype</b>	void pmu_privilege_disable(void);
<b>Function descriptions</b>	disable the privileged access only
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the privileged access only */
```

```
pmu_privilege_disable();
```

## pmu\_flag\_clear

The description of pmu\_flag\_clear is shown as below:

**Table 3-626. Function pmu\_flag\_clear**

<b>Function name</b>	pmu_flag_clear
<b>Function prototype</b>	void pmu_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear flag bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flag
<i>PMU_FLAG_RESET_WAKEUP</i>	reset wakeup flag
<i>PMU_FLAG_RESET_STANDBY</i>	reset standby flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-627. Function pmu\_flag\_get**

<b>Function name</b>	pmu_flag_get
<b>Function prototype</b>	FlagStatus pmu_flag_get(uint32_t pmu_flag);
<b>Function descriptions</b>	get flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pmu_flag</b>	flag
PMU_FLAG_WAKEUP	wakeup flag
PMU_FLAG_STANDBY	standby flag
PMU_FLAG_LVD	low voltage detector status flag
PMU_FLAG_VLVD	VDDA low voltage flag
PMU_FLAG_LDOVSRF	LDO voltage select ready flag
PMU_FLAG_LDRF	low-driver mode ready flag
PMU_FLAG_WIFI_SLEEP	WIFI is in sleep state
PMU_FLAG_WIFI_ACTIVE	WIFI is in active state
PMU_FLAG_SRAM1_SLEEP	SRAM1 is in sleep state
PMU_FLAG_SRAM1_ACTIVE	SRAM1 is in active state
PMU_FLAG_SRAM2_SLEEP	SRAM2 is in sleep state
PMU_FLAG_SRAM2_ACTIVE	SRAM2 is in active state
PMU_FLAG_SRAM3_SLEEP	SRAM3 is in sleep state
PMU_FLAG_SRAM3_ACTIVE	SRAM3 is in active state
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get flag state */
```

```
FlagStatus status;
```

```
status = pmu_flag_get(PMU_FLAG_WAKEUP);
```

## 3.21. QSPI

The QSPI is a specialized interface that communicate with Flash memories. This interface support single, dual or quad SPI FLASH. The QSPI registers are listed in chapter [3.21.1](#), the QSPI firmware functions are introduced in chapter [3.21.2](#).

### 3.21.1. Descriptions of Peripheral registers

QSPI registers are listed in the table shown as below:

**Table 3-628. QSPI registers**

Registers	Descriptions
QSPI_CTL	QSPI control register
QSPI_DCFG	QSPI device configuration register
QSPI_STAT	QSPI status register
QSPI_STATC	QSPI status clear register
QSPI_DTLEN	QSPI data length register
QSPI_TCFG	QSPI transfer configuration register
QSPI_ADDR	QSPI address register
QSPI_ALTE	QSPI alternate bytes register
QSPI_DATA	QSPI data register
QSPI_STATMK	QSPI status mask register
QSPI_STATMATCH	QSPI status match register
QSPI_INTERVAL	QSPI interval register
QSPI_TMOUT	QSPI timeout register
QSPI_FLUSH	QSPI fifo flush register
QSPI_WTCNT	QSPI wait cnt for indirect wire mode register
QSPI_STAT_SEC	QSPI secure status register
QSPI_STATC_SEC	QSPI secure status clear register
QSPI_DTLEN_SEC	QSPI secure data length register
QSPI_TCFG_SEC	QSPI secure transfer configuration register
QSPI_ADDR_SEC	QSPI secure address register
QSPI_ALTE_SEC	QSPI secure alternate bytes register
QSPI_DATA_SEC	QSPI secure data register

### 3.21.2. Descriptions of Peripheral functions

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

**Table 3-629. QSPI firmware function**

Function name	Function description
qspi_deinit	reset QSPI peripheral
qspi_init_struct_para_init	initialize the parameters of QSPI init struct with the default

Function name	Function description
	values
qspi_init	initialize QSPI peripheral parameter
qspi_enable	enable QSPI
qspi_disable	disable QSPI
qspi_dma_enable	enable QSPI DMA function
qspi_dma_disable	disable QSPI DMA function
qspi_command	QSPI command parameter configure
qspi_transmit	QSPI transmit data
qspi_receive	QSPI receive data
qspi_autopolling	configure QSPI autopolling mode
qspi_memorymapped	configure QSPI memorymapped mode
qspi_abort	abort the current transmission
qspi_interrupt_enable	enable QSPI interrupt
qspi_interrupt_disable	disable QSPI interrupt
qspi_flag_get	get QSPI flag status
qspi_flag_clear	clear QSPI flag status

### Structure qspi\_init\_struct

Table 3-630. qspi\_init\_struct

Member name	Function description
prescaler	QSPI prescaler (0x00 - 0xFF)
fifothreshold	QSPI FIFO threshold (0x01 - 0x1F)
sampleshift	QSPI sample shift (QSPI_SAMPLE_SHIFTING_NONE, QSPI_SAMPLE_SHIFTING_HALFCYCLE, QSPI_SAMPLE_SHIFTING_ONECYCLE)
flashsize	external flash size (0x00 - 0x1F)
chipselecthightime	CLK cycles which the chip select (nCS) must stay high between two command sequences (QSPI_CS_HIGH_TIME_n_CYCLE (n=1,2,...,8))
clockmode	QSPI clock mode (QSPI_CLOCK_MODE_0, QSPI_CLOCK_MODE_3)

### Structure qspi\_command\_struct

Table 3-631. qspi\_command\_struct

Member name	Function description
instructionmode	instruction mode (QSPI_INSTRUCTION_NONE, QSPI_INSTRUCTION_1_LINE,

	QSPI_INSTRUCTION_2_LINES, QSPI_INSTRUCTION_4_LINES)
instruction	8 bits instruction (0x00-0xFF)
addressmode	address mode (QSPI_ADDRESS_NONE, QSPI_ADDRESS_1_LINE, QSPI_ADDRESS_2_LINES, QSPI_ADDRESS_4_LINES )
addresssize	address size (QSPI_ADDRESS_8_BITS, QSPI_ADDRESS_16_BITS, QSPI_ADDRESS_24_BITS, QSPI_ADDRESS_32_BITS )
address	address to be send to the external Flash memory
alternatebytemode	alternate bytes mode (QSPI_ALTERNATE_BYTES_NONE, QSPI_ALTERNATE_BYTES_1_LINE, QSPI_ALTERNATE_BYTES_2_LINES, QSPI_ALTERNATE_BYTES_4_LINES)
addresssize	alternate bytes size (QSPI_ADDRESS_8_BITS, QSPI_ADDRESS_16_BITS, QSPI_ADDRESS_24_BITS, QSPI_ADDRESS_32_BITS)
alternatebytes	alternate bytes information
dummycycles	dummy cycles (0x00 - 0x1F)
datamode	data mode (QSPI_DATA_NONE, QSPI_DATA_1_LINE, QSPI_DATA_2_LINES, QSPI_DATA_4_LINES)
nbdata	32 bits data length
sioomode	Send instruction only once mode (QSPI_SIOO_INST_EVERY_CMD, QSPI_SIOO_INST_ONLY_FIRST_CMD)

## Structure qspi\_autopolling\_struct

**Table 3-632. qspi\_autopolling\_struct**

Member name	Function description
match	match value (0x0-0xFFFFFFFF)
mask	mask value (0x0-0xFFFFFFFF)
interval	number of clock cycles between two read during automatic polling phases (0x0-0xFFFF)
stautsbytesize	the size of the status bytes received (0x01-0x04)
matchmode	method used for determining a match (QSPI_MATCH_MODE_AND, QSPI_MATCH_MODE_OR)



automaticstop	if automatic polling is stopped after a match (QSPI_AUTOMATIC_STOP_DISABLE, QSPI_AUTOMATIC_STOP_ENABLE)
---------------	--

## qspi\_deinit

The description of qspi\_deinit is shown as below:

**Table 3-633. Function qspi\_deinit**

<b>Function name</b>	qspi_deinit
<b>Function prototype</b>	void qspi_deinit(void);
<b>Function descriptions</b>	reset QSPI peripheral
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset QSPI */
qspi_deinit();
```

## qspi\_init\_struct\_para\_init

The description of qspi\_init\_struct\_para\_init is shown as below:

**Table 3-634. Function qspi\_init\_struct\_para\_init**

<b>Function name</b>	qspi_init_struct_para_init
<b>Function prototype</b>	void qspi_init_struct_para_init(qspi_init_struct* qspi_struct);
<b>Function descriptions</b>	Initialize the parameters of QSPI init struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
qspi_struct	QSPI init parameter struct, the structure members can refer to <a href="#">Table 3-630. qspi_init_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the parameters of QSPI */
```

```
qspi_parameter_struct qspi_init_struct;
```

```
qspi_struct_para_init(&qspi_init_struct);
```

## qspi\_init

The description of qspi\_init is shown as below:

**Table 3-635. Function qspi\_init**

<b>Function name</b>	qspi_init
<b>Function prototype</b>	void qspi_init(qspi_init_struct* qspi_struct);
<b>Function descriptions</b>	Initialize QSPI peripheral parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>qspi_struct</b>	QSPI parameter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-630. qspi_init_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize QSPI */

qspi_init_struct qspi_struct;

qspi_struct.clockmode = QSPI_CLOCK_MODE_0;

qspi_struct.prescaler = 3;

qspi_struct.fifothreshold = 16;

qspi_struct.sampleshift = QSPI_SAMPLE_SHIFTING_NONE;

qspi_struct.flashsize = 0x1F;

qspi_struct.chipselecthightime = QSPI_CS_HIGH_TIME_1_CYCLE;

qspi_init(&qspi_struct);
```

## qspi\_enable

The description of qspi\_enable is shown as below:

**Table 3-636. Function quad\_spi\_enable**

<b>Function name</b>	qspi_enable
<b>Function prototype</b>	void qspi_enable(void);
<b>Function descriptions</b>	Enable QSPI

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable QSPI */
```

```
qspi_enable();
```

### qspi\_disable

The description of qspi\_disable is shown as below:

**Table 3-637. Function qspi\_disable**

<b>Function name</b>	qspi_disable
<b>Function prototype</b>	void qspi_disable(void);
<b>Function descriptions</b>	Disable QSPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable QSPI */
```

```
qspi_disable();
```

### qspi\_dma\_enable

The description of qspi\_dma\_enable is shown as below:

**Table 3-638. Function qspi\_dma\_enable**

<b>Function name</b>	qspi_dma_enable
<b>Function prototype</b>	void qspi_dma_enable(void);
<b>Function descriptions</b>	Enable QSPI DMA function
<b>Precondition</b>	-
<b>The called functions</b>	-

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

Example:

```
/* enable QSPI DMA function */
```

```
qspi_dma_enable();
```

## qspi\_dma\_disable

The description of qspi\_dma\_disable is shown as below:

**Table 3-639. Function qspi\_dma\_disable**

Function name	qspi_dma_disable
Function prototype	void qspi_dma_disable(void);
Function descriptions	Disable QSPI DMA function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable QSPI DMA function */
```

```
qspi_dma_disable();
```

## qspi\_data\_length\_config

The description of qspi\_data\_length\_config is shown as below:

**Table 3-640. Function qspi\_data\_length\_config**

Function name	qspi_data_length_config
Function prototype	void qspi_data_length_config(uint32_t dtlen);
Function descriptions	configure QSPI data length
Precondition	-
The called functions	-
Input parameter{in}	
dtlen	data length (1 ~ 4294967295 (4G-1))

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure QSPI data length */
qspi_data_length_config(4096);
```

## qspi\_command

The description of qspi\_command is shown as below:

**Table 3-641. Function qspi\_command**

Function name	qspi_command
Function prototype	void qspi_command(qspi_command_struct* command_struct);
Function descriptions	send QSPI command
Precondition	-
The called functions	-
Input parameter{in}	
<b>command_struct</b>	QSPI command struct, the structure members can refer to <a href="#">Table 3-631. qspi_command_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* send QSPI command */
qspi_command_struct command;
command.instruction_mode = QSPI_INSTRUCTION_1_LINE;
command.instruction = RDID;
command.addressmode = QSPI_ADDRESS_NONE;
command.addr_size = QSPI_ADDRESS_SIZE_8_BITS;
command.address = 0;
command.alternatebytemode = QSPI_ALTERNATE_BYTES_1_LINE;
command.alternatebytessize = QSPI_ALTERNATE_BYTES_24_BITS;
command.alter = 0;
command.dummy_cycles = 0;
```

```
command.data_mode = QSPI_DATA_1_LINE;

command.data_length = 3;

command.Sioomode = QSPI_SIOO_INST_EVERY_CMD;

qspi_command(&command);
```

## qspi\_transmit

The description of qspi\_transmit is shown as below:

**Table 3-642. Function qspi\_transmit**

<b>Function name</b>	qspi_transmit
<b>Function prototype</b>	void qspi_transmit(uint8_t *tdata);
<b>Function descriptions</b>	QSPI transmit data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>tdata</b>	transmit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* QSPI transmit data */

qspi_transmit(&tdata);
```

## qspi\_receive

The description of qspi\_receive is shown as below:

**Table 3-643. Function qspi\_receive**

<b>Function name</b>	qspi_receive
<b>Function prototype</b>	void qspi_receive(uint8_t *rdata)
<b>Function descriptions</b>	QSPI receive data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rdata</b>	receive data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* QSPI receive data */

qspi_receive(&rdata);
```

### qspi\_autopolling

The description of qspi\_autopolling is shown as below:

**Table 3-644. Function qspi\_command**

<b>Function name</b>	qspi_autopolling
<b>Function prototype</b>	void qspi_autopolling(qspi_command_struct *cmd, qspi_autopolling_struct *cfg);
<b>Function descriptions</b>	configure QSPI autopolling mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmd</b>	QSPI command struct, the structure members can refer to <a href="#">Table 3-631. qspi_command_struct</a>
<b>Input parameter{in}</b>	
<b>cfg</b>	QSPI command struct, the structure members can refer to <a href="#">Table 3-632. qspi_autopolling_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* send QSPI command */

qspi_command_struct command;

qspi_autopolling_struct autopolling_cmd;

command.instruction_mode = QSPI_INSTRUCTION_1_LINE;

command.instruction = RDID;

command.addressmode = QSPI_ADDRESS_NONE;

command.addr_size = QSPI_ADDRESS_SIZE_8_BITS;

command.address = 0;

command.alternatebytemode = QSPI_ALTERNATE_BYTES_1_LINE;

command.alternatebytessize = QSPI_ALTERNATE_BYTES_24_BITS;

command.alter = 0;
```

```

command.dummy_cycles = 0;

command.data_mode = QSPI_DATA_1_LINE;

command.data_length = 3;

command.Sioomode = QSPI_SIOO_INST_EVERY_CMD;

autopolling_cmd.match          = 0x02;

autopolling_cmd.mask           = 0x02;

autopolling_cmd.matchmode      = QSPI_MATCH_MODE_AND;

autopolling_cmd.statusbytesize = 1;

autopolling_cmd.interval       = 0x10;

autopolling_cmd.automaticstop  = QSPI_AUTOMATIC_STOP_ENABLE;

qspi_autopolling(&command, &autopolling_cmd);

```

### qspi\_memorymapped

The description of qspi\_memorymapped is shown as below:

**Table 3-645. Function qspi\_memorymapped**

Function name	qspi_memorymapped
Function prototype	void qspi_memorymapped(qspi_command_struct *cmd, uint16_t timeout, uint32_t toen);
Function descriptions	configure QSPI memorymapped mode
Precondition	-
The called functions	-
Input parameter{in}	
cmd	QSPI command struct, the structure members can refer to <a href="#">Table 3-631. qspi_command_struct</a>
Input parameter{in}	
timeout	QSPI command timeout value (0-0xFFFF)
Input parameter{in}	
toen	Timeout counter enable
QSPI_TOEN_DISABLE	Timeout counter disable
QSPI_TOEN_ENABLE	Timeout counter enable
Output parameter{out}	
-	-
Return value	
-	-

Example:



```

/* config QSPI memorymapped */

qspi_command_struct command;

command.instruction_mode = QSPI_INSTRUCTION_1_LINE;

command.instruction = RDID;

command.addressmode = QSPI_ADDRESS_NONE;

command.addr_size = QSPI_ADDRESS_SIZE_8_BITS;

command.address = 0;

command.alternatebytemode = QSPI_ALTERNATE_BYTES_1_LINE;

command.alternatebytessize = QSPI_ALTERNATE_BYTES_24_BITS;

command.alter = 0;

command.dummy_cycles = 0;

command.data_mode = QSPI_DATA_1_LINE;

command.data_length = 3;

command.Sioomode = QSPI_SIOO_INST_EVERY_CMD;

qspi_memorymapped (&command,0xff, QSPI_TOEN_ENABLE);

```

## qspi\_abort

The description of qspi\_abort is shown as below:

**Table 3-646. Function qspi\_abort**

<b>Function name</b>	qspi_abort
<b>Function prototype</b>	void qspi_abort(void);
<b>Function descriptions</b>	abort the current transmission
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* abort QSPI */

qspi_abort();

```

## qspi\_interrupt\_enable

The description of qspi\_interrupt\_enable is shown as below:

**Table 3-647. Function qspi\_interrupt\_enable**

<b>Function name</b>	qspi_interrupt_enable
<b>Function prototype</b>	void qspi_interrupt_enable( uint8_t interrupt);
<b>Function descriptions</b>	enable QSPI interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	QSPI interrupt
QSPI_INT_TC	transfer complete interrupt
QSPI_INT_FT	FIFO threshold interrupt
QSPI_INT_TERR	transfer error interrupt
QSPI_INT_SM	status match interrupt
QSPI_INT_TMOUT	timeout interrupt
QSPI_INT_WS	wrong start sequence interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* enable QSPI transfer complete interrupt */
```

```
qspi_interrupt_enable (QSPI_INT_TC);
```

## qspi\_interrupt\_disable

The description of qspi\_interrupt\_disable is shown as below:

**Table 3-648. Function qspi\_interrupt\_disable**

<b>Function name</b>	qspi_interrupt_disable
<b>Function prototype</b>	void qspi_interrupt_disable( uint8_t interrupt);
<b>Function descriptions</b>	disable QSPI interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	QSPI interrupt
QSPI_INT_TC	transfer complete interrupt
QSPI_INT_FT	FIFO threshold interrupt
QSPI_INT_TERR	transfer error interrupt
QSPI_INT_SM	status match interrupt
QSPI_INT_TMOUT	timeout interrupt

<i>QSPI_INT_WS</i>	wrong start sequence interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* disable QSPI transfer complete interrupt */
```

```
qspi_interrupt_disable (QSPI_INT_TC);
```

## qspi\_flag\_get

The description of qspi\_flag\_get is shown as below:

**Table 3-649. Function qspi\_flag\_get**

<b>Function name</b>	qspi_flag_get
<b>Function prototype</b>	FlagStatus qspi_flag_get(uint32_t flag);
<b>Function descriptions</b>	get QSPI flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	QSPI flag status
<i>QSPI_FLAG_TERR</i>	transfer error flag
<i>QSPI_FLAG_TC</i>	transfer complete flag
<i>QSPI_FLAG_FT</i>	FIFO threshold flag
<i>QSPI_FLAG_SM</i>	status match flag
<i>QSPI_FLAG_TMOUT</i>	timeout flag
<i>QSPI_FLAG_WS</i>	wrong start sequence flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get QSPI transfer complete flag */
```

```
while(RESET == qspi_flag_get (QSPI_FLAG_TC));
```

## qspi\_flag\_clear

The description of qspi\_flag\_clear is shown as below:

**Table 3-650. Function qspi\_flag\_clear**

<b>Function name</b>	qspi_flag_clear
<b>Function prototype</b>	void qspi_flag_clear( uint32_t flag);

<b>Function descriptions</b>	Clear QSPI flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	QSPI flag status
QSPI_FLAG_TERR	transfer error flag
QSPI_FLAG_TC	transfer complete flag
QSPI_FLAG_SM	status match flag
QSPI_FLAG_TMOUT	timeout flag
QSPI_FLAG_WS	wrong start sequence flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear QSPI transfer complete flag status */
```

```
qspi_flag_clear(QSPI_FLAG_TC);
```

## qspi\_config

The description of qspi\_config is shown as below:

**Table 3-651. Function qspi\_command**

<b>Function name</b>	qspi_config
<b>Function prototype</b>	static void qspi_config(qspi_command_struct *cmd, uint32_t functionalmode);
<b>Function descriptions</b>	configure QSPI functionalmode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmd</b>	QSPI command struct, the structure members can refer to <a href="#">Table 3-631. qspi_command_struct</a>
<b>Input parameter{in}</b>	
<b>functionalmode</b>	QSPI functionalmode select
QSPI_INDIRECT_WRITE	write operation in normal mode
QSPI_INDIRECT_READ	read operation in normal mode
QSPI_AUTO_POLLING	read operation in normal mode
QSPI_MEMORY_MAPPED	Memory map mode
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```

/* send QSPI command */

qspi_command_struct command;

qspi_autopolling_struct autopolling_cmd;

command.instruction_mode = QSPI_INSTRUCTION_1_LINE;

command.instruction = RDID;

command.addressmode = QSPI_ADDRESS_NONE;

command.addr_size = QSPI_ADDRESS_SIZE_8_BITS;

command.address = 0;

command.alternatebytemode = QSPI_ALTERNATE_BYTES_1_LINE;

command.alternatebytessize = QSPI_ALTERNATE_BYTES_24_BITS;

command.alter = 0;

command.dummy_cycles = 0;

command.data_mode = QSPI_DATA_1_LINE;

command.data_length = 3;

command.Sioomode = QSPI_SIOO_INST_EVERY_CMD;

qspi_config(&command, QSPI_MEMORY_MAPPED);

```

## 3.22. 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 [3.22.1](#), the RCU firmware functions are introduced in chapter [3.22.2](#).

### 3.22.1. Descriptions of Peripheral registers

**Table 3-652. RCU Registers**

Registers	Descriptions
RCU_CTL	control register
RCU_PLL	PLL register

Registers	Descriptions
RCU_CFG0	clock configuration register 0
RCU_INT	clock interrupt register
RCU_AHB1RST	AHB1 reset register
RCU_AHB2RST	AHB2 reset register
RCU_AHB3RST	AHB3 reset register
RCU_APB1RST	APB1 reset register
RCU_APB2RST	APB2 reset register
RCU_AHB1EN	AHB1 enable register
RCU_AHB2EN	AHB2 enable register
RCU_AHB3EN	AHB3 enable register
RCU_APB1EN	APB1 enable register
RCU_APB2EN	APB2 enable register
RCU_AHB1SPEN	AHB1 sleep mode enable register
RCU_AHB2SPEN	AHB2 sleep mode enable register
RCU_AHB3SPEN	AHB3 sleep mode enable register
RCU_APB1SPEN	APB1 sleep mode enable register
RCU_APB2SPEN	APB2 sleep mode enable register
RCU_BDCTL	backup domain control register
RCU_RSTSCK	reset source/clock register
RCU_PLLSSCTL	PLL clock spread spectrum control register
RCU_PLLCFG	PLL clock configuration register
RCU_CFG1	clock configuration register 1
RCU_ADDCTL	additional clock control register
RCU_SECP_CFG	secure configuration register
RCU_SECP_STAT	secure status register
RCU_AHB1SECP_STAT	AHB1 secure status register
RCU_AHB2SECP_STAT	AHB2 secure status register
RCU_AHB3SECP_STAT	AHB3 secure status register
RCU_APB1SECP_STAT	APB1 secure status register
RCU_APB2SECP_STAT	APB2 secure status register
RCU_VKEY	voltage key register
RCU_DSV	deep-sleep mode voltage register

### 3.22.2. Descriptions of Peripheral functions

**Table 3-653. RCU firmware function**

Function name	Function description
rcu_deinit	deinitialize the RCU
rcu_periph_clock_enable	enable the peripherals clock
rcu_periph_clock_disable	disable the peripherals clock
rcu_periph_clock_sleep_enable	enable the peripherals clock when in sleep mode

Function name	Function description
rcu_periph_clock_sleep_disable	disable the peripherals clock when in sleep mode
rcu_periph_reset_enable	enable the peripherals reset
rcu_periph_reset_disable	disable the peripheral reset
rcu_bkp_reset_enable	enable the BKP domain reset
rcu_bkp_reset_disable	disable the BKP domain reset
rcu_hxtal_plli2s_enable	enable HXTAL for PLLI2S
rcu_hxtal_plli2s_disable	disable HXTAL for PLLI2S
rcu_hxtal_pllp_enable	enable HXTAL for system CK_PLLP
rcu_hxtal_pllp_disable	disable HXTAL for system CK_PLLP
rcu_control_unit_powerup	power on the clock
rcu_control_unit_powerdown	power down the clock
rcu_rfppl_cal_enable	enable the RF PLL calculation
rcu_rfppl_cal_disable	disable the RF PLL calculation
rcu_system_clock_source_config	configure the system clock source
rcu_system_clock_source_get	get the system clock source
rcu_ahb_clock_config	configure the AHB clock prescaler selection
rcu_apb1_clock_config	configure the APB1 clock prescaler selection
rcu_apb2_clock_config	configure the APB2 clock prescaler selection
rcu_ckout0_config	configure the CK_OUT0 clock source
rcu_ckout1_config	configure the CK_OUT1 clock source and divider
rcu_pll_config	configure the main PLL clock
rcu_plli2s_config	configure the PLLI2S clock
rcu_plldig_config	configure the PLLDIG clock
rcu_plldig_div_sys_config	configure PLLDIG clock divider factor for system clock
rcu_rtc_clock_config	configure the RTC clock source selection
rcu_rtc_div_config	configure the frequency division of RTC clock when HXTAL was selected as its clock source
rcu_i2s_clock_config	configure the I2S clock source selection
rcu_pll_div_i2s_config	configure the PLL divider factor for I2S clock
rcu_hpdpf_clock_config	configure the HPDPF clock source selection (not support on GD32W515Tx series devices)
rcu_hpdpf_audio_clock_config	configure the HPDPF AUDIO clock source selection(not support on GD32W515Tx series devices)
rcu_sdio_clock_config	configure the SDIO clock source selection
rcu_sdio_div_config	configure the frequency division of the sdio source clock
rcu_usbfs_clock_config	configure the usb clock source selection
rcu_usbfs_div_config	configure the frequency division of the usbfs source clock
rcu_i2c0_clock_config	configure the I2C0 clock source selection
rcu_usart0_clock_config	configure the USART0 clock source selection
rcu_usart2_clock_config	configure the USART2 clock source selection
rcu_irc16m_div_config	configure IRC16M clock divider factor for system clock

Function name	Function description
rcu_timer_clock_prescaler_config	configure the TIMER clock prescaler selection
rcu_lxtal_drive_capability_config	configure the LXTAL drive capability
rcu_osci_stab_wait	wait for oscillator stabilization flags is SET or oscillator startup is timeout
rcu_osci_on	turn on the oscillator
rcu_osci_off	turn off the oscillator
rcu_osci_bypass_mode_enable	enable the oscillator bypass mode
rcu_osci_bypass_mode_disable	disable the oscillator bypass mode
rcu_irc16m_adjust_value_set	set the IRC16M adjust value
rcu_spread_spectrum_config	configure the spread spectrum modulation for the main PLL clock
rcu_spread_spectrum_enable	enable the spread spectrum modulation for the main PLL clock
rcu_spread_spectrum_disable	disable the spread spectrum modulation for the main PLL clock
rcu_hxtal_clock_monitor_enable	enable the HXTAL clock monitor
rcu_hxtal_clock_monitor_disable	disable the HXTAL clock monitor
rcu_rf_hxtal_clock_monitor_enable	enable the RF HXTAL clock monitor
rcu_rf_hxtal_clock_monitor_disable	disable the RF HXTAL clock monitor
rcu_voltage_key_unlock	unlock the voltage key
rcu_deepsleep_voltage_set	set the deep sleep mode voltage
rcu_clock_freq_get	get the system clock, bus clock frequency
rcu_flag_get	get the clock stabilization and peripheral reset flags
rcu_security_enable	enable the security attribution
rcu_security_disable	disable the security attribution
rcu_privilege_enable	enable the privileged access
rcu_privilege_disable	disable the privileged access
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_security_flag_get	get the secure status flag
rcu_interrupt_enable	enable the stabilization interrupt
rcu_interrupt_disable	disable the stabilization interrupt

## Enum rcu\_periph\_enum

Table 3-654. Enum rcu\_periph\_enum

enum name	Function description
RCU_GPIOA	GPIOA clock
RCU_GPIOB	GPIOB clock
RCU_GPIOC	GPIOC clock
RCU_TZPCU	TZPCU clock



## GD32W51x\_F5HC Firmware Library User Guide

enum name	Function description
RCU_TSI	TSI clock
RCU_CRC	CRC clock
RCU_WIFI	WIFI clock
RCU_WIFIRUN	WIFIRUN clock
RCU_SRAM0	SRAM0 clock
RCU_SRAM1	SRAM1 clock
RCU_SRAM2	SRAM2 clock
RCU_SRAM3	SRAM3 clock
RCU_DMA0	DMA0 clock
RCU_DMA1	DMA1 clock
RCU_USBFS	USBFS clock
RCU_DCI	DCI clock(DCI not support on GD32W515Tx series devices)
RCU_PKCAU	PKCAU clock
RCU_CAU	CAU clock
RCU_HAU	HAU clock
RCU_TRNG	TRNG clock
RCU_SQPI	SQPI clock
RCU_QSPI	QSPI clock
RCU_TIMER1	TIMER1 clock
RCU_TIMER2	TIMER2 clock
RCU_TIMER3	TIMER3 clock(TIMER3 not support on GD32W515Tx series devices)
RCU_TIMER4	TIMER4 clock
RCU_TIMER5	TIMER5 clock
RCU_WWDGT	WWDGT clock
RCU_SPI1	SPI1 clock
RCU_USART1	USART1 clock
RCU_USART0	USART0 clock
RCU_I2C0	I2C0 clock
RCU_I2C1	I2C1 clock
RCU_PMU	PMU clock
RCU_RTC	RTC clock
RCU_TIMER0	TIMER0 clock
RCU_USART2	USART0 clock
RCU_ADC	ADC clock
RCU_SDIO	SDIO clock
RCU_SPI0	SPI0 clock
RCU_SYSCFG	SYSCFG clock
RCU_TIMER15	TIMER15 clock
RCU_TIMER16	TIMER16 clock
RCU_HPDPF	HPDPF clock(HPDPF not support on GD32W515Tx series devices)
RCU_RF	RF clock

## Enum rcu\_periph\_sleep\_enum

**Table 3-655. Enum rcu\_periph\_sleep\_enum**

enum name	Function description
RCU_GPIOA_SLP	GPIOA clock when sleep mode
RCU_GPIOB_SLP	GPIOB clock when sleep mode
RCU_GPIOC_SLP	GPIOC clock when sleep mode
RCU_TZPCU_SLP	TZPCU clock when sleep mode
RCU_TSI_SLP	TSI clock when sleep mode
RCU_CRC_SLP	CRC clock when sleep mode
RCU_WIFI_SLP	WIFI clock when sleep mode
RCU_WIFIRUN_SLP	WIFIRUN clock when sleep mode
RCU_SRAM0_SLP	SRAM0 clock when sleep mode
RCU_SRAM1_SLP	SRAM1 clock when sleep mode
RCU_SRAM2_SLP	SRAM2 clock when sleep mode
RCU_SRAM3_SLP	SRAM3 clock when sleep mode
RCU_DMA0_SLP	DMA0 clock when sleep mode
RCU_DMA1_SLP	DMA1 clock when sleep mode
RCU_USBFS_SLP	USBFS clock when sleep mode
RCU_DCI_SLP	DCI clock when sleep mode(DCI not support on GD32W515Tx series devices)
RCU_PKCAU_SLP	PKCAU clock when sleep mode
RCU_CAU_SLP	CAU clock when sleep mode
RCU_HAU_SLP	HAU clock when sleep mode
RCU_TRNG_SLP	TRNG clock when sleep mode
RCU_SQPI_SLP	SQPI clock when sleep mode
RCU_QSPI_SLP	QSPI clock when sleep mode
RCU_TIMER1_SLP	TIMER1 clock when sleep mode
RCU_TIMER2_SLP	TIMER2 clock when sleep mode
RCU_TIMER3_SLP	TIMER3 clock when sleep mode(TIMER3 not support on GD32W515Tx series devices)
RCU_TIMER4_SLP	TIMER4 clock when sleep mode
RCU_TIMER5_SLP	TIMER5 clock when sleep mode
RCU_WWDGT_SLP	WWDGT clock when sleep mode
RCU_SPI1_SLP	SPI1 clock when sleep mode
RCU_USART1_SLP	USART1 clock when sleep mode
RCU_USART0_SLP	USART0 clock when sleep mode
RCU_I2C0_SLP	I2C0 clock when sleep mode
RCU_I2C1_SLP	I2C1 clock when sleep mode
RCU_PMU_SLP	PMU clock when sleep mode
RCU_RTC_SLP	RTC clock when sleep mode
RCU_TIMER0_SLP	TIMER0 clock when sleep mode
RCU_USART2_SLP	USART0 clock when sleep mode
RCU_ADC_SLP	ADC clock when sleep mode

enum name	Function description
RCU_SDIO_SLP	SDIO clock when sleep mode
RCU_SPI0_SLP	SPI0 clock when sleep mode
RCU_SYSCFG_SLP	SYSCFG clock when sleep mode
RCU_TIMER15_SLP	TIMER15 clock when sleep mode
RCU_TIMER16_SLP	TIMER16 clock when sleep mode
RCU_HPDF_SLP	HPDF clock when sleep mode(HPDF not support on GD32W515Tx series devices)
RCU_RF_SLP	RF clock when sleep mode

## Enum rcu\_periph\_reset\_enum

**Table 3-656. Enum rcu\_periph\_reset\_enum**

enum name	Function description
RCU_GPIOARST	GPIOA clock reset
RCU_GPIOBRST	GPIOB clock reset
RCU_GPIOCRST	GPIOC clock reset
RCU_TZPCURST	TZPCU clock reset
RCU_TSIRST	TSI clock reset
RCU_CRCRST	CRC clock reset
RCU_WIFIRST	WIFI clock reset
RCU_DMA0RST	DMA0 clock reset
RCU_DMA1RST	DMA1 clock reset
RCU_USBFIRST	USBFS clock reset
RCU_DCIRST	DCI clock reset(DCI not support on GD32W515Tx series devices)
RCU_PKCAURST	PKCAU clock reset
RCU_CAURST	CAU clock reset
RCU_HAURST	HAU clock reset
RCU_TRNGRST	TRNG clock reset
RCU_SQPIRST	SQPI clock reset
RCU_QSPIRST	QSPI clock reset
RCU_TIMER1RST	TIMER1 clock reset
RCU_TIMER2RST	TIMER2 clock reset
RCU_TIMER3RST	TIMER3 clock reset(TIMER3 not support on GD32W515Tx series devices)
RCU_TIMER4RST	TIMER4 clock reset
RCU_TIMER5RST	TIMER5 clock reset
RCU_WWDGTRST	WWDGT clock reset
RCU_SPI1RST	SPI1 clock reset
RCU_USART1RST	USART1 clock reset
RCU_USART0RST	USART0 clock reset
RCU_I2C0RST	I2C0 clock reset
RCU_I2C1RST	I2C1 clock reset
RCU_PMURST	PMU clock reset

enum name	Function description
RCU_RTCRST	RTC clock reset
RCU_TIMER0RST	TIMER0 clock reset
RCU_USART2RST	USART0 clock reset
RCU_ADCRST	ADC clock reset
RCU_SDIORST	SDIO clock reset
RCU_SPI0RST	SPI0 clock reset
RCU_SYSCFGRST	SYSCFG clock reset
RCU_TIMER15RST	TIMER15 clock reset
RCU_TIMER16RST	TIMER16 clock reset
RCU_HPDRST	HPDF clock reset(HPDF not support on GD32W515Tx series devices)
RCU_RFRST	RF clock reset

### Enum rcu\_flag\_enum

**Table 3-657. Enum rcu\_flag\_enum**

enum name	Function description
RCU_FLAG_IRC16MS TB	IRC16M stabilization flags
RCU_FLAG_HXTALST B	HXTAL stabilization flags
RCU_FLAG_PLLDIGST B	PLLDIG stabilization flags
RCU_FLAG_PLLSTB	PLL stabilization flags
RCU_FLAG_PLLI2SST B	PLLI2S stabilization flags
RCU_FLAG_LXTALST B	LXTAL stabilization flags
RCU_FLAG_IRC32KST B	IRC32K stabilization flags
RCU_FLAG_OBLRST	IRC48M stabilization flags
RCU_FLAG_EPRST	external PIN reset flags
RCU_FLAG_PORRST	power reset flags
RCU_FLAG_SWRST	software reset flags
RCU_FLAG_FWDGTR ST	FWDGT reset flags
RCU_FLAG_WWDGTR ST	WWDGT reset flags
RCU_FLAG_LPRST	low-power reset flags

## Enum rcu\_int\_flag\_enum

**Table 3-658. Enum rcu\_int\_flag\_enum**

enum name	Function description
RCU_INT_FLAG_IRC3 2KSTB	IRC32K stabilization interrupt flag
RCU_INT_FLAG_LXTA LSTB	LXTAL stabilization interrupt flag
RCU_INT_FLAG_IRC1 6MSTB	IRC16M stabilization interrupt flag
RCU_INT_FLAG_HXTA LSTB	HXTAL stabilization interrupt flag
RCU_INT_FLAG_PLLS TB	PLL stabilization interrupt flag
RCU_INT_FLAG_PLLI2 SSTB	PLLI2S stabilization interrupt flag
RCU_INT_FLAG_PLLD IGSTB	PLLDIG stabilization interrupt flag
RCU_INT_FLAG_CKM	HXTAL clock stuck interrupt flag

## Enum rcu\_int\_flag\_clear\_enum

**Table 3-659. Enum rcu\_int\_flag\_clear\_enum**

enum name	Function description
RCU_INT_FLAG_IRC3 2KSTB_CLR	IRC32K stabilization interrupt flags clear
RCU_INT_FLAG_LXTA LSTB_CLR	LXTAL stabilization interrupt flags clear
RCU_INT_FLAG_IRC1 6MSTB_CLR	IRC16M stabilization interrupt flags clear
RCU_INT_FLAG_HXTA LSTB_CLR	HXTAL stabilization interrupt flags clear
RCU_INT_FLAG_PLLS TB_CLR	PLL stabilization interrupt flags clear
RCU_INT_FLAG_PLLI2 SSTB_CLR	PLLI2S stabilization interrupt flags clear
RCU_INT_FLAG_PLLD IGSTB_CLR	PLLDIG stabilization interrupt flags clear
RCU_INT_FLAG_CKM _CLR	CKM interrupt flags clear

## Enum rcu\_int\_enum

**Table 3-660. Enum rcu\_int\_enum**

enum name	Function description
RCU_INT_IRC32KSTB	IRC32K stabilization interrupt
RCU_INT_LXTALSTB	LXTAL stabilization interrupt
RCU_INT_IRC16MSTB	IRC16M stabilization interrupt
RCU_INT_HXTALSTB	HXTAL stabilization interrupt
RCU_INT_PLLSTB	PLL stabilization interrupt
RCU_INT_PLLI2SSTB	PLLI2S stabilization interrupt
RCU_INT_PLLDIGSTB	PLLDIG stabilization interrupt

## Enum rcu\_osci\_type\_enum

**Table 3-661. Enum rcu\_osci\_type\_enum**

enum name	Function description
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL
RCU_IRC16M	IRC16M
RCU_IRC32K	IRC32K
RCU_PLLDIG_CK	PLLDIG
RCU_PLL_CK	PLL
RCU_PLLI2S_CK	PLLI2S

## Enum rcu\_clock\_freq\_enum

**Table 3-662. 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
CK_USART0	USART0 clock
CK_USART2	USART2 clock
CK_I2C0	I2C0 clock

## Enum rcu\_sec\_enum

**Table 3-663. Enum rcu\_sec\_enum**

enum name	Function description
RCU_SEC_IRC16MSEC	IRC16M clock configuration and status bits security
RCU_SEC_HXTALSEC	HXTAL clock configuration and status bits security
RCU_SEC_IRC32KSEC	IRC32K clock configuration and status bits security

enum name	Function description
C	
RCU_SEC_LXTALSEC	LXTAL clock configuration and status bits security
RCU_SEC_SYSCLOCKSEC	SYSCLK clock selection, STOPWUCK bit, clock output on MCO configuration
RCU_SEC_PRESCALERSEC	AHBx/APBx prescaler configuration bits security
RCU_SEC_PLLSEC	main PLL configuration and status bits security
RCU_SEC_PLLDIGSEC	PLLDIG configuration and status bits security
RCU_SEC_PLLI2SSEC	PLLI2S configuration and status bits security
RCU_SEC_RMVRSTFSEC	remove reset flag security
RCU_SEC_BKPSEC	BKP security

### Enum rcu\_sec\_flag\_enum

**Table 3-664. Enum rcu\_sec\_flag\_enum**

enum name	Function description
RCU_SEC_FLAG_IRC16MSEC	IRC16M clock configuration and status bits security
RCU_SEC_FLAG_HXTALSEC	HXTAL clock configuration and status bits security
RCU_SEC_FLAG_IRC32KSEC	IRC32K clock configuration and status bits security
RCU_SEC_FLAG_LXTALSEC	LXTAL clock configuration and status bits security
RCU_SEC_FLAG_SYSCLOCKSEC	SYSCLK clock selection, STOPWUCK bit, clock output on MCO configuration
RCU_SEC_FLAG_PRESCALERSEC	AHBx/APBx prescaler configuration bits security
RCU_SEC_FLAG_PLLSEC	main PLL configuration and status bits security
RCU_SEC_FLAG_PLLDIGSEC	PLLDIG configuration and status bits security
RCU_SEC_FLAG_PLLI2SSEC	PLLI2S configuration and status bits security
RCU_SEC_FLAG_RMVRSTFSEC	remove reset flag security
RCU_SEC_FLAG_BKPSEC	BKP security
RCU_SEC_FLAG_GPIOA	GPIOA clock

enum name	Function description
RCU_SEC_FLAG_GPIOB	GPIOB clock
RCU_SEC_FLAG_GPIOC	GPIOC clock
RCU_SEC_FLAG_CRC	CRC clock
RCU_SEC_FLAG_WIFI	WIFI clock
RCU_SEC_FLAG_FMC	FMC clock
RCU_SEC_FLAG_SRAM0	SRAM0 clock
RCU_SEC_FLAG_SRAM1	SRAM1 clock
RCU_SEC_FLAG_SRAM2	SRAM2 clock
RCU_SEC_FLAG_SRAM3	SRAM3 clock
RCU_SEC_FLAG_DMA0	DMA0 clock
RCU_SEC_FLAG_DMA1	DMA1 clock
RCU_SEC_FLAG_USBFS	USBFS clock
RCU_SEC_FLAG_DCI	DCI clock(DCI not support on GD32W515Tx series devices)
RCU_SEC_FLAG_PKCAU	PKCAU clock
RCU_SEC_FLAG_CAU	CAU clock
RCU_SEC_FLAG_HAU	HAU clock
RCU_SEC_FLAG_TRNG	TRNG clock
RCU_SEC_FLAG_SQPI	SQPI clock
RCU_SEC_FLAG_QSPI	QSPI clock
RCU_SEC_FLAG_TIMER1	TIMER1 clock
RCU_SEC_FLAG_TIMER2	TIMER2 clock
RCU_SEC_FLAG_TIMER3	TIMER3 clock
RCU_SEC_FLAG_TIMER4	TIMER4 clock
RCU_SEC_FLAG_TIMER5	TIMER5 clock



enum name	Function description
RCU_SEC_FLAG_WWDGT	WWDGT clock
RCU_SEC_FLAG_SPI1	SPI1 clock
RCU_SEC_FLAG_USART1	USART1 clock
RCU_SEC_FLAG_USART0	USART0 clock
RCU_SEC_FLAG_I2C0	I2C0 clock
RCU_SEC_FLAG_I2C1	I2C1 clock
RCU_SEC_FLAG_PMU	PMU clock
RCU_SEC_FLAG_TIMER0	TIMER0 clock
RCU_SEC_FLAG_USART2	USART2 clock
RCU_SEC_FLAG_ADC	ADC clock
RCU_SEC_FLAG_SDIO	SDIO clock
RCU_SEC_FLAG_SPI0	SPI0 clock
RCU_SEC_FLAG_SYSCFG	SYSCFG clock
RCU_SEC_FLAG_TIMER15	TIMER15 clock
RCU_SEC_FLAG_TIMER16	TIMER16 clock
RCU_SEC_FLAG_HPDA	HPDA clock(HPDA not support on GD32W515Tx series devices)
RCU_SEC_FLAG_RF	RF clock

## Enum rcu\_unit\_enum

**Table 3-665. Enum rcu\_unit\_enum**

enum name	Function description
RCU_UNIT_HXTAL	HXTAL
RCU_UNIT_PLLDIG	PLLDIG
RCU_UNIT_RFPLL	RFPLL
RCU_UNIT_LDOANA	LDOANA
RCU_UNIT_LDOCLK	LDOCLK
RCU_UNIT_BANDGAP	BANDGAP
RCU_UNIT_HXTAL	HXTAL

## rcu\_deinit

The description of rcu\_deinit is shown as below:

Table 3-666. Function rcu\_deinit

Function name	rcu_deinit
Function prototype	void rcu_deinit(void);
Function descriptions	deinitialize the RCU, reset the value of all RCU registers into initial values
Precondition	-
The called functions	rcu_oscstb_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset RCU */
rcu_deinit();
```

### rcu\_periph\_clock\_enable

The description of rcu\_periph\_clock\_enable is shown as below:

Table 3-667. 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 <a href="#">Table 3-654. Enum rcu_periph_enum</a>
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 3-668. Function rcu\_periph\_clock\_disable

Function name	rcu_periph_clock_disable
---------------	--------------------------

<b>Function prototype</b>	void rcu_periph_clock_disable(rcu_periph_enum periph);
<b>Function descriptions</b>	disable the peripherals clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-654. Enum rcu_periph_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-669. Function rcu\_periph\_clock\_sleep\_enable**

<b>Function name</b>	rcu_periph_clock_sleep_enable
<b>Function prototype</b>	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
<b>Function descriptions</b>	enable the peripherals clock when in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-655. Enum rcu_periph_sleep_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-670. Function rcu\_periph\_clock\_sleep\_disable**

<b>Function name</b>	rcu_periph_clock_sleep_disable
<b>Function prototype</b>	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);
<b>Function descriptions</b>	disable the peripherals clock when in sleep mode

Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to <a href="#">Table 3-655. Enum rcu_periph_sleep_enum</a>
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 3-671. Function rcu\_periph\_reset\_enable**

Function name	rcu_periph_reset_enable
Function prototype	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
Function descriptions	enable the peripherals reset
Precondition	-
The called functions	-
Input parameter{in}	
periph_reset	RCU peripherals reset, refer to <a href="#">Table 3-656. Enum rcu_periph_reset_enum</a>
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 3-672. Function rcu\_periph\_reset\_disable**

Function name	rcu_periph_reset_disable
Function prototype	void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);
Function descriptions	disable the peripheral reset
Precondition	-

The called functions	-
Input parameter{in}	
periph_reset	RCU peripherals reset, refer to <a href="#">Table 3-656. Enum rcu_periph_reset_enum</a>
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 3-673. Function rcu\_bkp\_reset\_enable**

Function name	rcu_bkp_reset_enable
Function prototype	void rcu_bkp_reset_enable(void);
Function descriptions	enable the BKP domain reset
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the BKP domain */
rcu_bkp_reset_enable();
```

### rcu\_bkp\_reset\_disable

The description of rcu\_bkp\_reset\_disable is shown as below:

**Table 3-674. 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\_hxtal\_plli2s\_enable

The description of rcu\_hxtal\_plli2s\_enable is shown as below:

**Table 3-675. Function rcu\_hxtal\_plli2s\_enable**

Function name	rcu_hxtal_plli2s_enable
Function prototype	void rcu_hxtal_plli2s_enable(void);
Function descriptions	enable HXTAL for PLLI2S
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable HXTAL for PLLI2S */
```

```
rcu_hxtal_plli2s_enable();
```

### rcu\_hxtal\_plli2s\_disable

The description of rcu\_hxtal\_plli2s\_disable is shown as below:

**Table 3-676. Function rcu\_hxtal\_plli2s\_disable**

Function name	rcu_hxtal_plli2s_disable
Function prototype	void rcu_hxtal_plli2s_disable(void);
Function descriptions	disable HXTAL for PLLI2S
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable HXTAL for PLLI2S */
rcu_hxtal_plli2s_disable();
```

## rcu\_hxtal\_pllp\_enable

The description of rcu\_hxtal\_pllp\_enable is shown as below:

**Table 3-677. Function rcu\_hxtal\_pllp\_enable**

Function name	rcu_hxtal_pllp_enable
Function prototype	void rcu_hxtal_pllp_enable(void);
Function descriptions	enable HXTAL for system CK_PLLP
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable HXTAL for system CK_PLLP */
rcu_hxtal_pllp_enable();
```

## rcu\_hxtal\_pllp\_disable

The description of rcu\_hxtal\_pllp\_disable is shown as below:

**Table 3-678. Function rcu\_hxtal\_pllp\_disable**

Function name	rcu_hxtal_pllp_disable
Function prototype	void rcu_hxtal_pllp_disable(void);
Function descriptions	disable HXTAL for system CK_PLLP
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable HXTAL for system CK_PLLP */
```

```
rcu_hxtal_pll_disable();
```

### rcu\_control\_unit\_powerup

The description of rcu\_control\_unit\_powerup is shown as below:

**Table 3-679. Function rcu\_control\_unit\_powerup**

Function name	rcu_control_unit_powerup
Function prototype	void rcu_control_unit_powerup (rcu_unit_enum rcu_unit);
Function descriptions	power on the clock
Precondition	-
The called functions	-
Input parameter{in}	
rcu_unit	
RCU_UNIT_HXTAL	power on the HXTAL
RCU_UNIT_PLLDIG	power on the PLLDIG
RCU_UNIT_RFPLL	enable the RF PLL calculation
RCU_UNIT_LDOANA	power on LDO analog
RCU_UNIT_LDOCLK	power on the LDO clock
RCU_UNIT_BANDGAP	power on the BandGap
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* power on the HXTAL */
```

```
rcu_control_unit_powerup(RCU_UNIT_HXTAL);
```

### rcu\_control\_unit\_powerdown

The description of rcu\_control\_unit\_powerdown is shown as below:

**Table 3-680. Function rcu\_control\_unit\_powerdown**

Function name	rcu_control_unit_powerdown
Function prototype	void rcu_control_unit_powerdown (rcu_unit_enum rcu_unit);
Function descriptions	power down the clock
Precondition	-
The called functions	-



Input parameter{in}	
rcu_unit	
RCU_UNIT_HXTAL	power down the HXTAL
RCU_UNIT_PLLDIG	power down the PLLDIG
RCU_UNIT_RFPLL	disenable the RF PLL calculation
RCU_UNIT_LDOANA	power down LDO analog
RCU_UNIT_LDOCLK	power down the LDO clock
RCU_UNIT_BANDGAP	power down the BandGap
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* power on the HXTAL */
```

```
rcu_control_unit_powerdown(RCU_UNIT_HXTAL);
```

### rcu\_rfpll\_cal\_enable

The description of rcu\_rfpll\_cal\_enable is shown as below:

**Table 3-681. Function rcu\_rfpll\_cal\_enable**

Function name	rcu_rfpll_cal_enable
Function prototype	void rcu_rfpll_cal_enable(void);
Function descriptions	enable the RF PLL calculation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the RF PLL calculation */
```

```
rcu_rfpll_cal_enable ();
```

### rcu\_rfpll\_cal\_disable

The description of rcu\_rfpll\_cal\_disable is shown as below:

**Table 3-682. Function rcu\_rfpll\_cal\_disable**

Function name	rcu_rfpll_cal_disable
---------------	-----------------------

<b>Function prototype</b>	void rcu_rfppl_cal_disable(void);
<b>Function descriptions</b>	disable the RF PLL calculation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the RF PLL calculation */
```

```
rcu_rfppl_cal_disable();
```

## rcu\_system\_clock\_source\_config

The description of rcu\_system\_clock\_source\_config is shown as below:

**Table 3-683. Function rcu\_system\_clock\_source\_config**

<b>Function name</b>	rcu_system_clock_source_config
<b>Function prototype</b>	void rcu_system_clock_source_config(uint32_t ck_sys);
<b>Function descriptions</b>	configure the system clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_sys</b>	system clock source select
<i>RCU_CKSYSSRC_IRC</i> <i>16M</i>	select CK_IRC16M as the CK_SYS source
<i>RCU_CKSYSSRC_HXTAL</i>	select CK_HXTAL as the CK_SYS source
<i>RCU_CKSYSSRC_PLL</i>	select CK_PLL as the CK_SYS source
<i>RCU_CKSYSSRC_PLL</i> <i>DIG</i>	select CK_PLLDIG as the CK_SYS source
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CK_HXTAL as the CK_SYS source */
```

```
rcu_system_clock_source_config(RCU_CKSYSSRC_HXTAL);
```

## rcu\_system\_clock\_source\_get

The description of rcu\_system\_clock\_source\_get is shown as below:

**Table 3-684. Function rcu\_system\_clock\_source\_get**

<b>Function name</b>	rcu_system_clock_source_get
<b>Function prototype</b>	uint32_t rcu_system_clock_source_get(void);
<b>Function descriptions</b>	get the system clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	RCU_SCSS_IRC16M/RCU_SCSS_HXTAL/RCU_SCSS_PLL/ RCU_CKSYSRC_PLLDIG

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 3-685. Function rcu\_ahb\_clock\_config**

<b>Function name</b>	rcu_ahb_clock_config
<b>Function prototype</b>	void rcu_ahb_clock_config(uint32_t ck_ahb);
<b>Function descriptions</b>	configure the AHB clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
ck_ahb	AHB clock prescaler selection
RCU_AHB_CKSYS_DIVx	select CK_SYS / x, (x=1, 2, 4, 8, 16, 64, 128, 256, 512)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-686. Function rcu\_apb1\_clock\_config**

<b>Function name</b>	rcu_apb1_clock_config
<b>Function prototype</b>	void rcu_apb1_clock_config(uint32_t ck_apb1);
<b>Function descriptions</b>	configure the APB1 clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_apb1</b>	APB1 clock prescaler selection
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-687. Function rcu\_apb2\_clock\_config**

<b>Function name</b>	rcu_apb2_clock_config
<b>Function prototype</b>	void rcu_apb2_clock_config(uint32_t ck_apb2);
<b>Function descriptions</b>	configure the APB2 clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_apb2</b>	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>IV2</i>	
<i>RCU_APB2_CK_AHB_D</i> <i>IV4</i>	select CK_AHB/4 as CK_APB2
<i>RCU_APB2_CK_AHB_D</i> <i>IV8</i>	select CK_AHB/8 as CK_APB2
<i>RCU_APB2_CK_AHB_D</i> <i>IV16</i>	select CK_AHB/16 as CK_APB2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure CK_AHB/8 as CK_APB2 */
```

```
rcu_apb2_clock_config(RCU_APB2_CK_AHB_DIV8);
```

## rcu\_ckout0\_config

The description of rcu\_ckout0\_config is shown as below:

**Table 3-688. Function rcu\_ckout0\_config**

<b>Function name</b>	rcu_ckout0_config
<b>Function prototype</b>	void rcu_ckout0_config(uint32_t ckout0_src);
<b>Function descriptions</b>	configure the CK_OUT0 clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ckout0_src</b>	CK_OUT0 clock source selection
<i>RCU_CKOUT0SRC_IR</i> <i>C16M</i>	select IRC16M
<i>RCU_CKOUT0SRC_LX</i> <i>TAL</i>	select LXTAL
<i>RCU_CKOUT0SRC_H</i> <i>XTAL</i>	select HXTAL
<i>RCU_CKOUT0SRC_PL</i> <i>LP</i>	select PLLP
<b>Input parameter{in}</b>	
<b>ckout0_div</b>	CK_OUT0 divider
<i>RCU_CKOUT0_DIVx</i>	CK_OUT0 is divided by x(x=1,2,3,4,5)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the HXTAL as CK_OUT0 clock source */

rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL, RCU_CKOUT0_DIV1);
```

### rcu\_ckout1\_config

The description of rcu\_ckout1\_config is shown as below:

**Table 3-689. Function rcu\_ckout1\_config**

<b>Function name</b>	rcu_ckout1_config
<b>Function prototype</b>	void rcu_ckout1_config(uint32_t ckout1_src);
<b>Function descriptions</b>	configure the CK_OUT1 clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ckout1_src</b>	CK_OUT1 clock source selection
<i>RCU_CKOUT1SRC_CKSYS</i>	select system clock
<i>RCU_CKOUT1SRC_PLI2S</i>	select PLLI2S
<i>RCU_CKOUT1SRC_HXTAL</i>	select HXTAL
<i>RCU_CKOUT1SRC_PLLDIG</i>	select PLLDIG
<b>Input parameter{in}</b>	
<b>ckout1_div</b>	CK_OUT1 divider
<i>RCU_CKOUT1_DIVx</i>	CK_OUT1 is divided by x(x=1,2,3,4,5)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the HXTAL as CK_OUT1 clock source */

rcu_ckout1_config(RCU_CKOUT1SRC_HXTAL, RCU_CKOUT1_DIV1);
```

### rcu\_pll\_config

The description of rcu\_pll\_config is shown as below:

**Table 3-690. Function rcu\_pll\_config**

<b>Function name</b>	rcu_pll_config
<b>Function prototype</b>	ErrStatus rcu_pll_config(uint32_t pll_src, uint32_t pll_psc, uint32_t pll_n,

	uint32_t pll_p);
<b>Function descriptions</b>	configure the main PLL clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_src</b>	PLL clock source selection
<i>RCU_PLLSRC_IRC16M</i>	IRC16M clock is selected as source clock of PLL
<i>RCU_PLLSRC_HXTAL</i>	HXTAL is selected as source clock of PLL
<b>Input parameter{in}</b>	
<b>pll_psc</b>	the PLL VCO source clock prescaler
<i>uint32_t</i>	2~63
<b>Input parameter{in}</b>	
<b>pll_n</b>	the PLL VCO clock multi factor
<i>uint32_t</i>	64~511
<b>Input parameter{in}</b>	
<b>pll_p</b>	the PLLP output frequency division factor from PLL VCO clock
<i>uint32_t</i>	2,4,6,8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* Configure the main PLL, PSC = 8, PLL_N = 240, PLL_P = 2 */
```

```
rcu_pll_config(RCU_PLLSRC_HXTAL,8,240,2);
```

## rcu\_plli2s\_config

The description of rcu\_plli2s\_config is shown as below:

**Table 3-691. Function rcu\_plli2s\_config**

<b>Function name</b>	rcu_plli2s_config
<b>Function prototype</b>	ErrStatus rcu_plli2s_config(uint32_t plli2s_n, uint32_t plli2s_psc,uint32_t plli2s_div);
<b>Function descriptions</b>	configure the PLLI2S clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>plli2s_n</b>	PLLI2S clock VCO clock multiplication factor
<i>uint32_t</i>	8~127
<b>Input parameter{in}</b>	
<b>plli2s_psc</b>	PLLI2S VCO source clock pre-scale

<i>RCU_PLLI2SSRC_DIV</i> x	PLLI2S VCO source clock is divided x(x=1,2,...,8)
<b>Input parameter{in}</b>	
<b>plli2s_div</b>	PLLI2SDIV clock divider factor
<i>RCU_PLLI2S_DIVx</i>	PLLI2SDIV clock is divided x(x=1.5,2,2.5,3,...,32)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* configure the I2S PLL, PLLI2S_PSC = 5, PLL_N = 10, PLLI2S_DIV = 2 */
```

```
rcu_plli2s_config (10, RCU_PLLI2SSRC_DIV5, RCU_PLLI2S_DIV2);
```

## rcu\_plldig\_config

The description of rcu\_plldig\_config is shown as below:

**Table 3-692. Function rcu\_pllpresel\_config**

<b>Function name</b>	rcu_plldig_config
<b>Function prototype</b>	void rcu_plldig_config(uint32_t plldig_clk)
<b>Function descriptions</b>	configure the PLLDIG output clock frequency
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>plldig_clk</b>	the PLLDIG VCO clock this parameter sselection
<i>RCU_PLLDIG_192M</i>	selected 192Mhz as PLLDIG output frequency
<i>RCU_PLLDIG_240M</i>	selected 240Mhz as PLLDIG output frequency
<i>RCU_PLLDIG_320M</i>	selected 320Mhz as PLLDIG output frequency
<i>RCU_PLLDIG_480M</i>	selected 480Mhz as PLLDIG output frequency
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* configure the PLLDIG output 192Mhz clock frequency */
```

```
rcu_plldig_config(RCU_PLLDIG_192M);
```

## rcu\_plldig\_div\_sys\_config

The description of rcu\_plldig\_div\_sys\_config is shown as below:



**Table 3-693. Function rcu\_plldig\_div\_sys\_config**

<b>Function name</b>	rcu_plldig_div_sys_config
<b>Function prototype</b>	void rcu_plldig_div_sys_config(uint32_t plldig_div_sys);
<b>Function descriptions</b>	configure PLLDIG clock divider factor for system clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>plldig_div_sys</b>	PLLDIG clock divider factor for system clock
<i>RCU_PLLDIG_SYS_DIVx</i>	PLLDIG clock divided by x for system clock(x=1,2,3,...,64)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* configure PLLDIG clock divider 2 for system clock */
```

```
rcu_plldig_div_sys_config (RCU_PLLDIG_SYS_DIV2);
```

## rcu\_rtc\_clock\_config

The description of rcu\_rtc\_clock\_config is shown as below:

**Table 3-694. Function rcu\_rtc\_clock\_config**

<b>Function name</b>	rcu_rtc_clock_config
<b>Function prototype</b>	void rcu_rtc_clock_config(uint32_t rtc_clock_source);
<b>Function descriptions</b>	configure the RTC clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_clock_source</b>	RTC clock source selection
<i>RCU_RTC_SRC_NONE</i>	no clock selected
<i>RCU_RTC_SRC_LXTAL</i>	CK_LXTAL selected as RTC source clock
<i>RCU_RTC_SRC_IRC32K</i>	CK_IRC32K selected as RTC source clock
<i>RCU_RTC_SRC_HXTAL_DIV_RTCDIV</i>	CK_HXTAL/RTCDIV selected as RTC source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select IRC32K as RTC clock source */
```

```
rcu_rtc_clock_config(RCU_RTCSRC_IRC32K);
```

### rcu\_rtc\_div\_config

The description of rcu\_rtc\_div\_config is shown as below:

**Table 3-695. Function rcu\_rtc\_div\_config**

<b>Function name</b>	rcu_rtc_div_config
<b>Function prototype</b>	void rcu_rtc_div_config(uint32_t rtc_div);
<b>Function descriptions</b>	configure the frequency division of RTC clock when HXTAL was selected as its clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_div</b>	RTC clock frequency division
<i>RCU_RTC_HXTAL_NONE</i>	no clock for RTC
<i>RCU_RTC_HXTAL_DIVx</i>	RTCDIV clock select CK_HXTAL/x, x = 2....31
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* RTCDIV clock select CK_HXTAL/2 */
```

```
rcu_rtc_div_config(RCU_RTC_HXTAL_DIV2);
```

### rcu\_i2s\_clock\_config

The description of rcu\_i2s\_clock\_config is shown as below:

**Table 3-696. Function rcu\_i2s\_clock\_config**

<b>Function name</b>	rcu_i2s_clock_config
<b>Function prototype</b>	void rcu_i2s_clock_config(uint32_t i2s_clock_source);
<b>Function descriptions</b>	configure the I2S clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2s_clock_source</b>	I2S clock source selection
<i>RCU_I2SSRC_PLLI2S</i>	CK_PLLI2S selected as I2S source clock
<i>RCU_I2SSRC_I2S_CKIN</i>	external i2s_ckin pin selected as I2S source clock

<i>RCU_I2SSRC_I2S_PL</i> <i>LDIV</i>	PLL division selected as I2S source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select CK_PLLI2S as I2S source clock */
rcu_i2s_clock_config(RCU_I2SSRC_PLLI2S);
```

### rcu\_pll\_div\_i2s\_config

The description of rcu\_pll\_div\_i2s\_config is shown as below:

**Table 3-697. Function rcu\_pll\_div\_i2s\_config**

<b>Function name</b>	rcu_pll_div_i2s_config
<b>Function prototype</b>	void rcu_pll_div_i2s_config(uint32_t pll_div_i2s);
<b>Function descriptions</b>	configure the PLL divider factor for I2S clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_div_i2s</b>	PLL divider factor for I2S clock
<i>RCU_PLLDIV_I2S_DIV</i> <i>x</i>	PLL clock divided by x for I2S clock (x=1,2,...,64)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PLL clock divided by 8 for I2S clock */
rcu_pll_div_i2s_config (RCU_PLLDIV_I2S_DIV8);
```

### rcu\_hpdf\_clock\_config

The description of rcu\_hpdf\_clock\_config is shown as below:

**Table 3-698. Function rcu\_hpdf\_clock\_config**

<b>Function name</b>	rcu_hpdf_clock_config
<b>Function prototype</b>	void rcu_hpdf_clock_config(uint32_t hpdf_clock_source);
<b>Function descriptions</b>	configure the hpdf clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>hpdf_clock_source</b>	hpdf clock source selection
<i>RCU_HPDFSRC_PCLK2</i>	PCLK2 clock selected as HPDF source clock
<i>RCU_HPDFSRC_CKSYS</i>	system clock selected as HPDF source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select system clock as hpdf clock source */
```

```
rcu_hpdf_clock_config(RCU_HPDFSRC_CKSYS);
```

### rcu\_hpdf\_audio\_clock\_config

The description of rcu\_hpdf\_audio\_clock\_config is shown as below:

**Table 3-699. Function rcu\_hpdf\_audio\_clock\_config**

<b>Function name</b>	rcu_hpdf_audio_clock_config
<b>Function prototype</b>	void rcu_hpdf_clock_config(uint32_t hpdf_clock_source);
<b>Function descriptions</b>	configure the HPDF AUDIO clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>hpdfaudio_clock_source</b>	HPDF AUDIO clock source selection
<i>RCU_HPDAUDIOSRC_PLLI2S</i>	PLLI2S output clock selected as HPDF AUDIO source clock
<i>RCU_HPDAUDIOSRC_I2S_CKIN</i>	external I2S_CKIN PIN selected as HPDF AUDIO source clock
<i>RCU_HPDAUDIOSRC_PLL</i>	PLL division selected as HPDF AUDIO source clock
<i>RCU_HPDAUDIOSRC_IRC16M</i>	IRC16M selected as HPDF AUDIO source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select PLLI2S clock as hpdf audio clock source */
```

```
rcu_hpdf_audio_clock_config(RCU_HPDAUDIOSRC_PLLI2S);
```

## rcu\_sdio\_clock\_config

The description of rcu\_sdio\_clock\_config is shown as below:

**Table 3-700. Function rcu\_sdio\_clock\_config**

<b>Function name</b>	rcu_sdio_clock_config
<b>Function prototype</b>	void rcu_sdio_clock_config(uint32_t sdio_clock_source);
<b>Function descriptions</b>	SDIO clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sdio_clock_source</b>	SDIO clock source selection
<i>RCU_SDIOSRC_PLL</i>	PLL output clock selected as SDIO source clock
<i>RCU_SDIOSRC_PLLDIG</i>	PLLDIG selected as SDIO source clock
<i>RCU_SDIOSRC_IRC16M</i>	IRC16M selected as SDIO source clock
<i>RCU_SDIOSRC_HXTAL</i>	HXTAL selected as SDIO source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select PLL clock as sdio clock source */
rcu_sdio_clock_config(RCU_SDIOSRC_PLL);
```

## rcu\_sdio\_div\_config

The description of rcu\_sdio\_div\_config is shown as below:

**Table 3-701. Function rcu\_sdio\_div\_config**

<b>Function name</b>	rcu_sdio_div_config
<b>Function prototype</b>	void rcu_sdio_div_config(uint32_t sdio_div);
<b>Function descriptions</b>	configure the frequency division of the sdio source clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sdio_div</b>	SDIO clock frequency division
<i>RCU_SDIODIV_DIVx</i>	SDIODIV input source clock divided by x,(x = 1....32)
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* SDIODIV input source clock divided by 4 */
```

```
rcu_sdio_div_config(RCU_SDIODIV_DIV4);
```

### rcu\_usbfs\_clock\_config

The description of rcu\_usbfs\_clock\_config is shown as below:

**Table 3-702. Function rcu\_usbfs\_clock\_config**

<b>Function name</b>	rcu_usbfs_clock_config
<b>Function prototype</b>	void rcu_usbfs_clock_config(uint32_t usbfs_clock_source);
<b>Function descriptions</b>	configure the USBFS clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usbfs_clock_source</b>	USBFS clock source selection
<i>RCU_USBFSRC_PLL</i>	PLL clock selected as USB source clock
<i>RCU_USBFSRC_PLL DIG</i>	PLLDIG clock selected as USB source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PLL clock selected as USB source clock */
```

```
rcu_usbfs_clock_config(RCU_USBFSRC_PLL);
```

### rcu\_usbfs\_div\_config

The description of rcu\_usbfs\_div\_config is shown as below:

**Table 3-703. Function rcu\_usbfs\_div\_config**

<b>Function name</b>	rcu_usbfs_div_config
<b>Function prototype</b>	void rcu_usbfs_div_config(uint32_t usbfs_div);
<b>Function descriptions</b>	configure the frequency division of the usbfs source clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usbfs_div</b>	USBFS clock frequency division

<i>RCU_USBFS_DIVx</i>	USBFS DIV input source clock divided by x, (x = 1,...,32)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USBFS DIV input source clock divided by 4 */
```

```
rcu_usbfs_div_config(RCU_USBFS_DIV4);
```

## rcu\_i2c0\_clock\_config

The description of rcu\_i2c0\_clock\_config is shown as below:

**Table 3-704. Function rcu\_i2c0\_clock\_config**

<b>Function name</b>	rcu_i2c0_clock_config
<b>Function prototype</b>	void rcu_i2c0_clock_config(uint32_t i2c0_clock_source);
<b>Function descriptions</b>	configure the I2C0 clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c0_clock_source</b>	I2C0 clock source selection
<i>RCU_I2C0SRC_CKAPB1</i>	CK_APB1 selected as I2C0 source clock
<i>RCU_I2C0SRC_CKSYS</i>	CK_SYS selected as I2C0 source clock
<i>RCU_I2C0SRC_IRC16M</i>	CK_IRC16M selected as I2C0 source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select IRC16M as I2C0 source clock */
```

```
rcu_i2c0_clock_config(RCU_I2C0SRC_IRC16M);
```

## rcu\_usart0\_clock\_config

The description of rcu\_usart0\_clock\_config is shown as below:

**Table 3-705. Function rcu\_usart0\_clock\_config**

<b>Function name</b>	rcu_usart0_clock_config
<b>Function prototype</b>	void rcu_usart0_clock_config(uint32_t usart0_clock_source);

<b>Function descriptions</b>	configure the USART0 clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart0_clock_source</b>	USART0 clock source selection
<i>RCU_USART0SRC_CKAPB1</i>	CK_APB1 selected as USART0 source clock
<i>RCU_USART0SRC_CKSYS</i>	CK_SYS selected as USART0 source clock
<i>RCU_USART0SRC_CK_LXTAL</i>	CK_LXTAL selected as USART0 source clock
<i>RCU_USART0SRC_CK_IRC16M</i>	CK_IRC16M selected as USART0 source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select APB1 clock as USART0 clock */
```

```
rcu_usart0_clock_config(RCU_USART0SRC_CKAPB1);
```

### rcu\_usart2\_clock\_config

The description of rcu\_usart2\_clock\_config is shown as below:

**Table 3-706. Function rcu\_usart2\_clock\_config**

<b>Function name</b>	rcu_usart2_clock_config
<b>Function prototype</b>	void rcu_usart2_clock_config(uint32_t usart2_clock_source);
<b>Function descriptions</b>	configure the USART2 clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart2_clock_source</b>	USART2 clock source selection
<i>RCU_USART2SRC_CKAPB1</i>	CK_APB1 selected as USART2 source clock
<i>RCU_USART2SRC_CKSYS</i>	CK_SYS selected as USART2 source clock
<i>RCU_USART2SRC_CK_LXTAL</i>	CK_LXTAL selected as USART2 source clock
<i>RCU_USART2SRC_CK_IRC16M</i>	CK_IRC16M selected as USART2 source clock
<b>Output parameter{out}</b>	



-	-
Return value	
-	-

Example:

```
/* select CK_APB1 as USART2 source clock */
```

```
rcu_usart2_clock_config(RCU_USART2SRC_CKAPB1);
```

### rcu\_irc16m\_div\_config

The description of rcu\_irc16m\_div\_config is shown as below:

**Table 3-707. Function rcu\_irc16m\_div\_config**

<b>Function name</b>	rcu_irc16m_div_config
<b>Function prototype</b>	void rcu_irc16m_div_config(uint32_t irc16m_div);
<b>Function descriptions</b>	configure IRC16M clock divider factor for system clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>irc16m_div</b>	IRC16M clock divider factor for system clock
<i>RCU_IRC16M_DIVx</i>	IRC16M clock divided by x for system clock (x=1,2,3,...,512)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* IRC16M clock divided by 4 for system clock */
```

```
rcu_irc16m_div_config(RCU_IRC16M_DIV4);
```

### rcu\_timer\_clock\_prescaler\_config

The description of rcu\_timer\_clock\_prescaler\_config is shown as below:

**Table 3-708. Function rcu\_sdio\_div\_config**

<b>Function name</b>	rcu_timer_clock_prescaler_config
<b>Function prototype</b>	void rcu_timer_clock_prescaler_config(uint32_t timer_clock_prescaler);
<b>Function descriptions</b>	configure the TIMER clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_clock_prescaler</b>	TIMER clock selection
<b>RCU_TIMER_PSC_MU L2</b>	If CK_APBx = CK_AHB or CK_APBx = CK_AHB/2, CK_TIMERx = CK_AHB, else CK_TIMERx = 2 x CK_APBx

<b>RCU_TIMER_PSC_MUL4</b>	If CK_APBx = CK_AHB or CK_APBx = CK_AHB/2 or CK_APBx = CK_AHB/4, CK_TIMERx = CK_AHB, else CK_TIMERx = 4 x CK_APBx
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER clock source */
```

```
rcu_timer_clock_prescaler_config(RCU_TIMER_PSC_MUL4);
```

### rcu\_lxtal\_drive\_capability\_config

The description of rcu\_lxtal\_drive\_capability\_config is shown as below:

**Table 3-709. Function rcu\_lxtal\_drive\_capability\_config**

<b>Function name</b>	rcu_lxtal_drive_capability_config
<b>Function prototype</b>	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
<b>Function descriptions</b>	configure the LXTAL drive capability
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lxtal_dricap</b>	drive capability of LXTAL
<i>RCU_LXTALDRI_LOWER_DRIVE</i>	lower driving capability
<i>RCU_LXTALDRI_HIGH_DRIVE</i>	high driving capability
<i>RCU_LXTALDRI_HIGHER_DRIVE</i>	higher driving capability
<i>RCU_LXTALDRI_HIGHEST_DRIVE</i>	highest driving capability
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the LXTAL drive capability */
```

```
rcu_lxtal_drive_capability_config (RCU_LXTAL_LOWDRI);
```

### rcu\_osc\_stab\_wait

The description of rcu\_osc\_stab\_wait is shown as below:

**Table 3-710. Function rcu\_osci\_stab\_wait**

<b>Function name</b>	rcu_osci_stab_wait
<b>Function prototype</b>	ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci);
<b>Function descriptions</b>	wait for oscillator stabilization flags is SET or oscillator startup is timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-661. Enum rcu_osci_type_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* wait for oscillator stabilization flag */
if(SUCCESS == rcu_osci_stab_wait(RCU_HXTAL)){
}
```

## rcu\_osci\_on

The description of rcu\_osci\_on is shown as below:

**Table 3-711. Function rcu\_osci\_on**

<b>Function name</b>	rcu_osci_on
<b>Function prototype</b>	void rcu_osci_on(rcu_osci_type_enum osci);
<b>Function descriptions</b>	turn on the oscillator
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-661. Enum rcu_osci_type_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn on the high speed crystal oscillator */
rcu_osci_on(RCU_HXTAL);
```

## rcu\_osci\_off

The description of rcu\_osci\_off is shown as below:

**Table 3-712. Function rcu\_osci\_off**

<b>Function name</b>	rcu_osci_off
<b>Function prototype</b>	void rcu_osci_off(rcu_osci_type_enum osci);
<b>Function descriptions</b>	turn off the oscillator
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-661. Enum rcu_osci_type_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn off the high speed crystal oscillator */
```

```
rcu_osci_off(RCU_HXTAL);
```

## rcu\_osci\_bypass\_mode\_enable

The description of rcu\_osci\_bypass\_mode\_enable is shown as below:

**Table 3-713. Function rcu\_osci\_bypass\_mode\_enable**

<b>Function name</b>	rcu_osci_bypass_mode_enable
<b>Function prototype</b>	void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);
<b>Function descriptions</b>	enable the oscillator bypass mode
<b>Precondition</b>	HXTALEN or LXTALEN must be reset before it
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-661. Enum rcu_osci_type_enum</a>
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

## rcu\_osci\_bypass\_mode\_disable

The description of rcu\_osci\_bypass\_mode\_disable is shown as below:

**Table 3-714. Function rcu\_osci\_bypass\_mode\_disable**

<b>Function name</b>	rcu_osci_bypass_mode_disable
<b>Function prototype</b>	void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci);
<b>Function descriptions</b>	disable the oscillator bypass mode
<b>Precondition</b>	HXTALEN or LXTALEN must be reset before it
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-661. Enum rcu_osci_type_enum</a>
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the high speed crystal oscillator bypass mode */
rcu_osci_bypass_mode_disable(RCU_HXTAL);
```

## rcu\_hxtal\_clock\_monitor\_enable

The description of rcu\_hxtal\_clock\_monitor\_enable is shown as below:

**Table 3-715. Function rcu\_hxtal\_clock\_monitor\_enable**

<b>Function name</b>	rcu_hxtal_clock_monitor_enable
<b>Function prototype</b>	void rcu_hxtal_clock_monitor_enable(void);
<b>Function descriptions</b>	enable the HXTAL clock monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-716. Function rcu\_hxtal\_clock\_monitor\_disable**

<b>Function name</b>	rcu_hxtal_clock_monitor_disable
<b>Function prototype</b>	void rcu_hxtal_clock_monitor_disable(void);
<b>Function descriptions</b>	disable the HXTAL clock monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the HXTAL clock monitor */
rcu_hxtal_clock_monitor_disable();
```

## rcu\_rf\_hxtal\_clock\_monitor\_enable

The description of rcu\_rf\_hxtal\_clock\_monitor\_enable is shown as below:

**Table 3-717. Function rcu\_rf\_hxtal\_clock\_monitor\_enable**

<b>Function name</b>	rcu_rf_hxtal_clock_monitor_enable
<b>Function prototype</b>	void rcu_rf_hxtal_clock_monitor_enable(void);
<b>Function descriptions</b>	enable the RF HXTAL clock monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the RF HXTAL clock monitor */
rcu_rf_hxtal_clock_monitor_enable();
```

## rcu\_rf\_hxtal\_clock\_monitor\_disable

The description of rcu\_rf\_hxtal\_clock\_monitor\_disable is shown as below:

**Table 3-718. Function rcu\_rf\_hxtal\_clock\_monitor\_disable**

<b>Function name</b>	rcu_rf_hxtal_clock_monitor_disable
----------------------	------------------------------------

<b>Function prototype</b>	void rcu_rf_hxtal_clock_monitor_disable(void);
<b>Function descriptions</b>	disable the RF HXTAL clock monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the RF HXTAL clock monitor */
rcu_rf_hxtal_clock_monitor_disable();
```

## rcu\_IRC16M\_adjust\_value\_set

The description of rcu\_IRC16M\_adjust\_value\_set is shown as below:

**Table 3-719. Function rcu\_IRC16M\_adjust\_value\_set**

<b>Function name</b>	rcu_IRC16M_adjust_value_set
<b>Function prototype</b>	void rcu_IRC16M_adjust_value_set(uint32_t IRC16M_adjval);
<b>Function descriptions</b>	set the IRC16M adjust value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>IRC16M_adjval</b>	IRC16M adjust value, must be between 0 and 0x1F
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the IRC16M adjust value */
rcu_IRC16M_adjust_value_set(0x10);
```

## rcu\_spread\_spectrum\_config

The description of rcu\_spread\_spectrum\_config is shown as below:

**Table 3-720. Function rcu\_spread\_spectrum\_config**

<b>Function name</b>	rcu_spread_spectrum_config
<b>Function prototype</b>	void rcu_spread_spectrum_config(uint32_t spread_spectrum_type, uint32_t modstep, uint32_t modcnt);

<b>Function descriptions</b>	configure the spread spectrum modulation for the main PLL clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>spread_spectrum_type</i>	PLL spread spectrum modulation type select
<i>RCU_SS_TYPE_CENTER</i>	center spread type is selected
<i>RCU_SS_TYPE_DOWN</i>	down spread type is selected
<b>Input parameter{in}</b>	
<b>modstep</b>	configure PLL spread spectrum modulation profile amplitude
<i>uint32_t</i>	0 ~ 0x7FFF, The following criteria must be met: $MODSTEP * MODCNT \leq 2^{15}-1$
<b>Input parameter{in}</b>	
<b>modcnt</b>	configure PLL spread spectrum modulation profile frequency
<i>uint32_t</i>	0 ~ 0x1FFF, The following criteria must be met: $MODSTEP * MODCNT \leq 2^{15}-1$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure PLL spread_spectrum */
```

```
rcu_spread_spectrum_config (RCU_SS_TYPE_CENTER, 0x0F,0x0F);
```

## rcu\_spread\_spectrum\_enable

The description of rcu\_spread\_spectrum\_enable is shown as below:

**Table 3-721. Function rcu\_spread\_spectrum\_enable**

<b>Function name</b>	rcu_spread_spectrum_enable
<b>Function prototype</b>	void rcu_spread_spectrum_enable(void);
<b>Function descriptions</b>	enable the PLL spread spectrum modulation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:



```
/* enable the PLL spread spectrum modulation */
```

```
rcu_spread_spectrum_enable ();
```

### rcu\_spread\_spectrum\_disable

The description of rcu\_spread\_spectrum\_disable is shown as below:

**Table 3-722. Function rcu\_spread\_spectrum\_disable**

<b>Function name</b>	rcu_spread_spectrum_disable
<b>Function prototype</b>	void rcu_spread_spectrum_disable(void);
<b>Function descriptions</b>	disable the HXTAL clock monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the PLL spread spectrum modulation */
```

```
rcu_spread_spectrum_disable();
```

### rcu\_voltage\_key\_unlock

The description of rcu\_voltage\_key\_unlock is shown as below:

**Table 3-723. Function rcu\_voltage\_key\_unlock**

<b>Function name</b>	rcu_voltage_key_unlock
<b>Function prototype</b>	void rcu_voltage_key_unlock(void);
<b>Function descriptions</b>	unlock the voltage key
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*unlock the voltage key */
```

```
rcu_voltage_key_unlock();
```

## rcu\_deepsleep\_voltage\_set

The description of rcu\_deepsleep\_voltage\_set is shown as below:

**Table 3-724. Function rcu\_deepsleep\_voltage\_set**

<b>Function name</b>	rcu_deepsleep_voltage_set
<b>Function prototype</b>	void rcu_deepsleep_voltage_set(uint32_t dsvol);
<b>Function descriptions</b>	set the deep-sleep mode voltage value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dsvol</b>	deep sleep mode voltage
RCU_DEEPSLEEP_V_1_1	the core voltage is 1.1V in deep-sleep mode
RCU_DEEPSLEEP_V_1_0	the core voltage is 1.0V in deep-sleep mode
RCU_DEEPSLEEP_V_0_9	the core voltage is 0.9V in deep-sleep mode
RCU_DEEPSLEEP_V_0_8	the core voltage is 0.8V in deep-sleep mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-725. Function rcu\_clock\_freq\_get**

<b>Function name</b>	rcu_clock_freq_get
<b>Function prototype</b>	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
<b>Function descriptions</b>	get the system clock, bus clock frequency
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>clock</b>	Clock frequency, refers to <a href="#">Table 3-662. Enum rcu_clock_freq_enum</a>
<b>Output parameter{out}</b>	

-	-
<b>Return value</b>	
<b>ck_freq</b>	clock frequency of system, AHB, APB1, APB2

Example:

```
uint32_t temp_freq;

/* get the system clock frequency */

temp_freq = rcu_clock_freq_get(CK_SYS);
```

## rcu\_security\_enable

The description of rcu\_security\_enable is shown as below:

**Table 3-726. Function rcu\_security\_enable**

<b>Function name</b>	rcu_security_enable
<b>Function prototype</b>	void rcu_security_enable(rcu_sec_enum security);
<b>Function descriptions</b>	enable the security attribution
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>security</b>	clock security attribution, refer to <a href="#">Table 3-663. Enum rcu_sec_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the PLLI2S configuration and status bits security attribution */

rcu_security_enable (RCU_SEC_PLLI2SSEC);
```

## rcu\_security\_disable

The description of rcu\_security\_disable is shown as below:

**Table 3-727. Function rcu\_security\_disable**

<b>Function name</b>	rcu_security_disable
<b>Function prototype</b>	void rcu_security_disable(rcu_sec_enum security);
<b>Function descriptions</b>	disable the security attribution
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>security</b>	clock security attribution, refer to <a href="#">Table 3-663. Enum rcu_sec_enum</a>
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* disable the PLLI2S configuration and status bits security attribution */
```

```
rcu_security_disable (RCU_SEC_PLLI2SSEC);
```

## rcu\_privilege\_enable

The description of rcu\_privilege\_enable is shown as below:

**Table 3-728. Function rcu\_privilege\_enable**

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

Example:

```
/* privileged access enable */
```

```
rcu_privilege_enable ();
```

## rcu\_privilege\_disable

The description of rcu\_privilege\_disable is shown as below:

**Table 3-729. Function rcu\_privilege\_disable**

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

-	-
---	---

Example:

```
/* disable the privileged access */
```

```
rcu_privilege_disable ();
```

## rcu\_flag\_get

The description of rcu\_flag\_get is shown as below:

**Table 3-730. Function rcu\_flag\_get**

<b>Function name</b>	rcu_flag_get
<b>Function prototype</b>	FlagStatus rcu_flag_get(rcu_flag_enum flag);
<b>Function descriptions</b>	get the clock stabilization and peripheral reset flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
flag	the clock stabilization and peripheral reset flags, refer to <a href="#">Table 3-657. Enum rcu_flag_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-731. Function rcu\_all\_reset\_flag\_clear**

<b>Function name</b>	rcu_all_reset_flag_clear
<b>Function prototype</b>	void rcu_all_reset_flag_clear(void);
<b>Function descriptions</b>	clear all the reset flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

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 3-732. Function rcu\_interrupt\_flag\_get**

<b>Function name</b>	rcu_interrupt_flag_get
<b>Function prototype</b>	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);
<b>Function descriptions</b>	get the clock stabilization interrupt and ckm flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt and ckm flags, refer to <a href="#">Table 3-657. Enum rcu_flag_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-733. Function rcu\_interrupt\_flag\_clear**

<b>Function name</b>	rcu_interrupt_flag_clear
<b>Function prototype</b>	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag);
<b>Function descriptions</b>	clear the interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	clock stabilization and stuck interrupt flags clear, refer to <a href="#">Table 3-659. Enum rcu_int_flag_clear_enum</a>
<b>Output parameter{out}</b>	

Example:

```
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

## rcu\_security\_flag\_get

The description of `rcu_security_flag_get` is shown as below:

### Table 3-734. Function rcu\_security\_flag\_get

491

### Table 3-735. Function rcu\_interrupt\_enable

```
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```



## rcu\_interrupt\_disable

The description of rcu\_interrupt\_disable is shown as below:

**Table 3-736. Function rcu\_interrupt\_disable**

<b>Function name</b>	rcu_interrupt_disable
<b>Function prototype</b>	void rcu_interrupt_disable(rcu_int_enum interrupt);
<b>Function descriptions</b>	disable the stabilization interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	clock stabilization interrupt, refer to <a href="#">Table 3-660. Enum rcu_int_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

## 3.23. 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, two alarms, a prescaler, a divider and the RTC clock configuration register. The RTC registers are listed in chapter [3.23.1](#), the RTC firmware functions are introduced in chapter [3.23.2](#).

### 3.23.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

**Table 3-737. RTC Registers**

Registers	Descriptions
RTC_TIME	time of day register
RTC_DATE	date register
RTC_CTL	control register
RTC_ICS	initialization control and status register
RTC_PSC	time prescaler register
RTC_WUT	wakeup timer register
RTC_COSC	coarse calibration register
RTC_ALRM0TD	alarm 0 time and date register
RTC_ALRM1TD	alarm 1 time and date register

Registers	Descriptions
RTC_WPK	write protection key register
RTC_SS	sub second register
RTC_SHIFTCTL	shift function control register
RTC_TTS	time of timestamp register
RTC_DTS	date of timestamp register
RTC_SSTS	sub second of timestamp register
RTC_HRFC	high resolution frequency compensation register
RTC_TAMP	tamper register
RTC_ALRM0SS	alarm 0 sub second register
RTC_ALRM1SS	alarm1 sub second register
RTC_PPM_CTL	privilege protection mode control register
RTC_SPM_CTL	secure protection mode control register
RTC_STAT	status register
RTC_NSMI_STAT	non-secure masked interrupt status register
RTC_SMI_STAT	secure masked interrupt status register
RTC_STATC	status flag clear register
RTC_BKPx(x = 0, 1, 2, ..., 18, 19)	backup register

## 3.23.2. Descriptions of Peripheral functions

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

**Table 3-738. RTC firmware function**

Function name	Function description
rtc_deinit	reset most of the RTC registers
rtc_init	initialize RTC registers
rtc_init_mode_enter	enter RTC init mode
rtc_init_mode_exit	exit RTC init mode
rtc_register_sync_wait	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
rtc_current_time_get	get current time and date
rtc_subsecond_get	get current subsecond value
rtc_alarm_config	configure RTC alarm
rtc_alarm_subsecond_config	configure subsecond of RTC alarm
rtc_alarm_get	get RTC alarm
rtc_alarm_subsecond_get	get RTC alarm subsecond
rtc_alarm_enable	enable RTC alarm
rtc_alarm_disable	disable RTC alarm
rtc_timestamp_enable	enable RTC time-stamp
rtc_timestamp_disable	disable RTC time-stamp

Function name	Function description
rtc_timestamp_get	get RTC timestamp time and date
rtc_timestamp_subsecond_get	get RTC time-stamp subsecond
rtc_tamper_enable	enable RTC tamper
rtc_tamper_disable	disable RTC tamper
rtc_software_bkp_reset	reset the RTC_BKP registers by software
rtc_tamper_without_bkp_seset	tamperx event does not erase the RTC_BKP registers
rtc_output_pin_select	select the RTC output pin pad
rtc_alarm_output_config	configure RTC alarm output source
rtc_calibration_output_config	configure RTC calibration output source
rtc_hour_adjust	ajust the daylight saving time by adding or subtracting one hour from the current time
rtc_second_adjust	ajust RTC second or subsecond value of current time
rtc_bypass_shadow_enable	enable RTC bypass shadow registers function
rtc_bypass_shadow_disable	disable RTC bypass shadow registers function
rtc_refclock_detection_enable	enable RTC reference clock detection function
rtc_refclock_detection_disable	disable RTC reference clock detection function
rtc_wakeup_enable	enable RTC wakeup timer
rtc_wakeup_disable	disable RTC wakeup timer
rtc_wakeup_clock_set	set auto wakeup timer clock
rtc_wakeup_timer_set	set auto wakeup timer value
rtc_wakeup_timer_get	get auto wakeup timer value
rtc_smooth_calibration_config	configure RTC smooth calibration
rtc_coarse_calibration_enable	enable RTC coarse calibration
rtc_coarse_calibration_disable	disable RTC coarse calibration
rtc_coarse_calibration_config	configure RTC coarse calibration direction and step
rtc_pri_pro_enable	enable RTC privilege protection mode
rtc_pri_pro_disable	disable RTC privilege protection mode
rtc_sec_pro_enable	enable RTC secure protection mode
rtc_sec_pro_disable	disable RTC secure protection mode
rtc_bkp_zonea_sec_pro_set	set the RTC_BKP secure protection zonea tail value
rtc_bkp_zoneb_sec_pro_set	set the RTC_BKP secure protection zoneb tail value
rtc_bkp_zoneb_sec_pro_check	check the RTC_BKP secure protection zoneb is valid or not
rtc_flag_get	get specified flag
rtc_flag_clear	clear specified flag
rtc_interrupt_enable	enable specified RTC interrupt
rtc_interrupt_disable	disble specified RTC interrupt
rtc_nsec_interrupt_flag_get	get specified non-secure interrupt flag
rtc_nsec_interrupt_flag_clear	clear specified non-secure interrupt flag
rtc_sec_interrupt_flag_get	get specified secure interrupt flag
rtc_sec_interrupt_flag_clear	clear specified secure interrupt flag

## Structure rtc\_parameter\_struct

**Table 3-739. Structure rtc\_parameter\_struct**

Member name	Function description
year	RTC year value: 0x0 - 0x99(BCD format)
month	RTC month value
date	RTC date value: 0x1 - 0x31(BCD format)
day_of_week	RTC weekday value
hour	RTC hour value
minute	RTC minute value: 0x0 - 0x59(BCD format)
second	RTC second value: 0x0 - 0x59(BCD format)
factor_asyn	RTC asynchronous prescaler value: 0x0 - 0x7F
factor_syn	RTC synchronous prescaler value: 0x0 - 0x7FFF
am_pm	RTC AM/PM value
display_format	RTC time notation

## Structure rtc\_alarm\_struct

**Table 3-740. Structure rtc\_alarm\_struct**

Member name	Function description
alarm_mask	RTC alarm mask
weekday_or_date	specify RTC alarm is on date or weekday
alarm_day	RTC alarm date or weekday value
alarm_hour	RTC alarm hour value
alarm_minute	RTC alarm minute value: 0x0 - 0x59(BCD format)
alarm_second	RTC alarm second value: 0x0 - 0x59(BCD format)
am_pm	RTC alarm AM/PM value

## Structure rtc\_timestamp\_struct

**Table 3-741. Structure rtc\_timestamp\_struct**

Member name	Function description
timestamp_month	RTC time-stamp month value
timestamp_date	RTC time-stamp date value: 0x1 - 0x31(BCD format)
timestamp_day	RTC time-stamp weekday value
timestamp_hour	RTC time-stamp hour value
timestamp_minute	RTC time-stamp minute value: 0x0 - 0x59(BCD format)
timestamp_second	RTC time-stamp second value: 0x0 - 0x59(BCD format)
am_pm	RTC time-stamp AM/PM value

## Structure rtc\_tamper\_struct

**Table 3-742. Structure rtc\_tamper\_struct**

Member name	Function description
-------------	----------------------

tamper_source	RTC tamper source
tamper_trigger	RTC tamper trigger
tamper_filter	RTC tamper consecutive samples needed during a voltage level detection
tamper_sample_frequency	RTC tamper sampling frequency during a voltage level detection
tamper_precharge_enable	RTC tamper precharge feature during a voltage level detection
tamper_precharge_time	RTC tamper precharge duration if precharge feature is enabled
tamper_with_timestamp	RTC tamper time-stamp feature

### rtc\_deinit

The description of rtc\_deinit is shown as below:

**Table 3-743. Function rtc\_deinit**

<b>Function name</b>	rtc_deinit
<b>Function prototype</b>	ErrStatus rtc_deinit(void);
<b>Function descriptions</b>	reset most of the RTC registers
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable/ rcu_periph_reset_disable -
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* reset most of the RTC registers*/
```

```
ErrStatus error_status = rtc_deinit();
```

### rtc\_init

The description of rtc\_init is shown as below:

**Table 3-744. Function rtc\_init**

<b>Function name</b>	rtc_init
<b>Function prototype</b>	ErrStatus rtc_init(rtc_parameter_struct* rtc_initpara_struct);
<b>Function descriptions</b>	initialize RTC registers
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>rtc_initpara_struct</b>	pointer to a rtc_parameter_struct structure which contains parameters for initialization of the rtc peripheral, the structure members can refer to members of the structure <a href="#">Table 3-739. Structure rtc_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```

/* initialize RTC registers */

rtc_parameter_struct rtc_initpara;

rtc_interrupt_disable(RTC_INT_SECOND);

rtc_initpara.factor_asyn = prescaler_a;

rtc_initpara.factor_syn = prescaler_s;

rtc_initpara.year = 0x16;

rtc_initpara.day_of_week = RTC_SATURDAY;

rtc_initpara.month = RTC_APR;

rtc_initpara.date = 0x30;

rtc_initpara.display_format = RTC_24HOUR;

rtc_initpara.am_pm = RTC_AM;

rtc_init(&rtc_initpara);

```

### rtc\_init\_mode\_enter

The description of rtc\_init\_mode\_enter is shown as below:

**Table 3-745. Function rtc\_init\_mode\_enter**

<b>Function name</b>	rtc_init_mode_enter
<b>Function prototype</b>	ErrStatus rtc_init_mode_enter(void);
<b>Function descriptions</b>	enter RTC init mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/*enter RTC init mode*/
```

```
ErrStatus error_status = rtc_init_mode_enter ();
```

## rtc\_init\_mode\_exit

The description of rtc\_init\_mode\_exit is shown as below:

**Table 3-746. Function rtc\_init\_mode\_exit**

<b>Function name</b>	rtc_init_mode_exit
<b>Function prototype</b>	void rtc_init_mode_exit(void);
<b>Function descriptions</b>	exit RTC init mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*exit RTC init mode*/
```

```
rtc_init_mode_exit ();
```

## rtc\_register\_sync\_wait

The description of rtc\_register\_sync\_wait is shown as below:

**Table 3-747. Function rtc\_register\_sync\_wait**

<b>Function name</b>	rtc_register_sync_wait
<b>Function prototype</b>	ErrStatus rtc_register_sync_wait(void);
<b>Function descriptions</b>	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/*wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the
shadow registers are updated*/
```

```
ErrStatus error_status = rtc_register_sync_wait ();
```

### rtc\_current\_time\_get

The description of rtc\_current\_time\_get is shown as below:

**Table 3-748. Function rtc\_current\_time\_get**

<b>Function name</b>	rtc_current_time_get
<b>Function prototype</b>	void rtc_current_time_get(rtc_parameter_struct* rtc_initpara_struct);
<b>Function descriptions</b>	get current time and date
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>rtc_initpara_struct</b>	pointer to a rtc_parameter_struct structure which contains parameters for initialization of the rtc peripheral, the structure members can refer to members of the structure <a href="#">Table 3-739. Structure rtc_parameter_struct</a>
<b>Return value</b>	
-	-

Example:

```
/*get current time and date*/

rtc_parameter_struct rtc_initpara_struct;

rtc_current_time_get (&rtc_initpara_struct);
```

### rtc\_subsecond\_get

The description of rtc\_subsecond\_get is shown as below:

**Table 3-749. Function rtc\_subsecond\_get**

<b>Function name</b>	rtc_subsecond_get
<b>Function prototype</b>	uint32_t rtc_subsecond_get(void);
<b>Function descriptions</b>	get current subsecond value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	



uint32_t	current subsecond value(0x00-0xFFFF)
----------	--------------------------------------

Example:

```
/*get current subsecond value*/
```

```
uint32_t sub_second = rtc_subsecond_get();
```

### rtc\_alarm\_config

The description of rtc\_alarm\_config is shown as below:

**Table 3-750. Function rtc\_alarm\_config**

<b>Function name</b>	rtc_alarm_config
<b>Function prototype</b>	void rtc_alarm_config(uint8_t rtc_alarm, rtc_alarm_struct* rtc_alarm_time)
<b>Function descriptions</b>	configure RTC alarm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_alarm</b>	select alarm
<i>RTC_ALARM0</i>	alarm 0
<i>RTC_ALARM1</i>	alarm 1
<b>Input parameter{in}</b>	
<b>rtc_alarm_time</b>	pointer to a rtc_alarm_struct structure which contains parameters for RTC alarm configuration, the structure members can refer to members of the structure <a href="#">Table 3-740. Structure rtc_alarm_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*rtc_alarm_config*/
```

```
rtc_alarm_struct rtc_alarm_time;
```

```
rtc_alarm_config(RTC_ALARM0,&rtc_alarm_time);
```

### rtc\_alarm\_subsecond\_config

The description of rtc\_alarm\_subsecond\_config is shown as below:

**Table 3-751. Function rtc\_alarm\_subsecond\_config**

<b>Function name</b>	rtc_alarm_subsecond_config
<b>Function prototype</b>	void rtc_alarm_subsecond_config(uint8_t rtc_alarm, uint32_t mask_subsecond, uint32_t subsecond)
<b>Function descriptions</b>	configure subsecond of RTC alarm

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_alarm</b>	select alarm
<i>RTC_ALARM0</i>	alarm 0
<i>RTC_ALARM1</i>	alarm 1
<b>Input parameter{in}</b>	
<b>mask_subsecond</b>	alarm subsecond mask
<i>RTC_MASKSSC_0_14</i>	mask alarm subsecond configuration
<i>RTC_MASKSSC_1_14</i>	mask RTC_ALRM0SS_SSC[14:1], and RTC_ALRM0SS_SSC[0] is to be compared
<i>RTC_MASKSSC_2_14</i>	mask RTC_ALRM0SS_SSC[14:2], and RTC_ALRM0SS_SSC[1:0] is to be compared
<i>RTC_MASKSSC_3_14</i>	mask RTC_ALRM0SS_SSC[14:3], and RTC_ALRM0SS_SSC[2:0] is to be compared
<i>RTC_MASKSSC_4_14</i>	mask RTC_ALRM0SS_SSC[14:4], and RTC_ALRM0SS_SSC[3:0] is to be compared
<i>RTC_MASKSSC_5_14</i>	mask RTC_ALRM0SS_SSC[14:5], and RTC_ALRM0SS_SSC[4:0] is to be compared
<i>RTC_MASKSSC_6_14</i>	mask RTC_ALRM0SS_SSC[14:6], and RTC_ALRM0SS_SSC[5:0] is to be compared
<i>RTC_MASKSSC_7_14</i>	mask RTC_ALRM0SS_SSC[14:7], and RTC_ALRM0SS_SSC[6:0] is to be compared
<i>RTC_MASKSSC_8_14</i>	mask RTC_ALRM0SS_SSC[14:8], and RTC_ALRM0SS_SSC[7:0] is to be compared
<i>RTC_MASKSSC_9_14</i>	mask RTC_ALRM0SS_SSC[14:9], and RTC_ALRM0SS_SSC[8:0] is to be compared
<i>RTC_MASKSSC_10_14</i>	mask RTC_ALRM0SS_SSC[14:10], and RTC_ALRM0SS_SSC[9:0] is to be compared
<i>RTC_MASKSSC_11_14</i>	mask RTC_ALRM0SS_SSC[14:11], and RTC_ALRM0SS_SSC[10:0] is to be compared
<i>RTC_MASKSSC_12_14</i>	mask RTC_ALRM0SS_SSC[14:12], and RTC_ALRM0SS_SSC[11:0] is to be compared
<i>RTC_MASKSSC_13_14</i>	mask RTC_ALRM0SS_SSC[14:13], and RTC_ALRM0SS_SSC[12:0] is to be compared
<i>RTC_MASKSSC_14</i>	mask RTC_ALRM0SS_SSC[14], and RTC_ALRM0SS_SSC[13:0] is to be compared
<i>RTC_MASKSSC_NONE</i>	mask none, and RTC_ALRM0SS_SSC[14:0] is to be compared
<b>Input parameter{in}</b>	
<b>subsecond</b>	alarm subsecond value(0x000 - 0x7FFF)
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/*configure subsecond of RTC alarm0*/
```

```
rtc_subsecond_config (RTC_ALARM0,RTC_MASKSSC_9_14, 0x7FFF);
```

## rtc\_alarm\_get

The description of rtc\_alarm\_get is shown as below:

**Table 3-752. Function rtc\_alarm\_get**

Function name	rtc_alarm_get
Function prototype	void rtc_alarm_get(uint8_t rtc_alarm, rtc_alarm_struct* rtc_alarm_time);
Function descriptions	get RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm
RTC_ALARM0	alarm 0
RTC_ALARM1	alarm 1
Output parameter{out}	
rtc_alarm_time	pointer to a rtc_alarm_struct structure which contains parameters for RTC alarm configuration, the structure members can refer to members of the structure <a href="#">Table 3-740. Structure rtc_alarm_struct</a>
Return value	
-	-

Example:

```
rtc_alarm_struct rtc_alarm_time;
```

```
rtc_alarm_get(RTC_ALARM0,&rtc_alarm_time);
```

## rtc\_alarm\_subsecond\_get

The description of rtc\_alarm\_subsecond\_get is shown as below:

**Table 3-753. Function rtc\_alarm\_subsecond\_get**

Function name	rtc_alarm_subsecond_get
Function prototype	uint32_t rtc_alarm_subsecond_get(uint8_t rtc_alarm)
Function descriptions	get RTC alarm subsecond
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm

<i>RTC_ALARM0</i>	alarm 0
<i>RTC_ALARM1</i>	alarm 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	RTC alarm subsecond value(0x0-0x7FFF)

Example:

```
/*get RTC alarm0 subsecond*/
```

```
uint32_t subsecond = rtc_alarm_subsecond_get(RTC_ALARM0);
```

### rtc\_alarm\_enable

The description of rtc\_alarm\_enable is shown as below:

**Table 3-754. Function rtc\_alarm\_enable**

<b>Function name</b>	rtc_alarm_enable
<b>Function prototype</b>	void rtc_alarm_enable(uint8_t rtc_alarm)
<b>Function descriptions</b>	enable RTC alarm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_alarm</b>	select alarm
<i>RTC_ALARM0</i>	alarm 0
<i>RTC_ALARM1</i>	alarm 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*enable RTC alarm0*/
```

```
rtc_alarm_enable(RTC_ALARM0);
```

### rtc\_alarm\_disable

The description of rtc\_alarm\_disable is shown as below:

**Table 3-755. Function rtc\_alarm\_disable**

<b>Function name</b>	rtc_alarm_disable
<b>Function prototype</b>	ErrStatus rtc_alarm_disable(uint8_t rtc_alarm)
<b>Function descriptions</b>	disable RTC alarm
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>rtc_alarm</b>	select alarm
<i>RTC_ALARM0</i>	alarm 0
<i>RTC_ALARM1</i>	alarm 1
Output parameter{out}	
-	-
Return value	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/*disable RTC alarm0*/
```

```
ErrStatus error_status = rtc_alarm_disable(RTC_ALARM0);
```

### rtc\_timestamp\_enable

The description of rtc\_timestamp\_enable is shown as below:

**Table 3-756. Function rtc\_timestamp\_enable**

<b>Function name</b>	rtc_timestamp_enable
<b>Function prototype</b>	void rtc_timestamp_enable(uint32_t edge);
<b>Function descriptions</b>	enable RTC time-stamp
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>edge</b>	specify which edge to detect of time-stamp
<i>RTC_TIMESTAMP_RISING_EDGE</i>	rising edge is valid event edge for timestamp event
<i>RTC_TIMESTAMP_FALLING_EDGE</i>	falling edge is valid event edge for timestamp event
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*enable RTC time-stamp*/
```

```
rtc_timestamp_enable (RTC_TIMESTAMP_RISING_EDGE);
```

### rtc\_timestamp\_disable

The description of rtc\_timestamp\_disable is shown as below:

**Table 3-757. Function rtc\_timestamp\_disable**

<b>Function name</b>	rtc_timestamp_disable
----------------------	-----------------------

<b>Function prototype</b>	void rtc_timestamp_disable(void);
<b>Function descriptions</b>	disable RTC time-stamp
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable RTC time-stamp */
rtc_timestamp_disable ();
```

### rtc\_timestamp\_get

The description of rtc\_timestamp\_get is shown as below:

**Table 3-758. Function rtc\_timestamp\_get**

<b>Function name</b>	rtc_timestamp_get
<b>Function prototype</b>	void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);
<b>Function descriptions</b>	get RTC timestamp time and date
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
rtc_timestamp	Pointer to a rtc_timestamp_struct structure which contains parameters for RTC time-stamp configuration, the structure members can refer to members of the structure <a href="#">Table 3-742. Structure rtc_tamper_struct</a>
<b>Return value</b>	
-	-

Example:

```
/* get RTC timestamp time and date */
rtc_timestamp_struct rtc_timestamp;
rtc_timestamp_get(& rtc_timestamp);
```

### rtc\_timestamp\_subsecond\_get

The description of rtc\_timestamp\_subsecond\_get is shown as below:

**Table 3-759. Function rtc\_timestamp\_subsecond\_get**

<b>Function name</b>	rtc_timestamp_subsecond_get
<b>Function prototype</b>	uint32_t rtc_timestamp_subsecond_get(void);
<b>Function descriptions</b>	get RTC time-stamp subsecond
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	RTC time-stamp subsecond value

Example:

```
/* get RTC time-stamp subsecond */
uint32_t subsecond = rtc_timestamp_subsecond_get();
```

## rtc\_tamper\_enable

The description of rtc\_tamper\_enable is shown as below:

**Table 3-760. Function rtc\_tamper\_enable**

<b>Function name</b>	rtc_tamper_enable
<b>Function prototype</b>	void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);
<b>Function descriptions</b>	enable RTC tamper
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
rtc_tamper	pointer to a rtc_tamper_struct structure which contains parameters for RTC tamper configuration, the structure members can refer to members of the structure <a href="#">Table 3-742. Structure rtc_tamper_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable RTC tamper */
rtc_tamper_struct rtc_tamper
rtc_tamper_enable(& rtc_tamper);
```

## rtc\_tamper\_disable

The description of rtc\_tamper\_disable is shown as below:

**Table 3-761. Function rtc\_tamper\_disable**

<b>Function name</b>	rtc_tamper_disable
<b>Function prototype</b>	void rtc_tamper_disable(uint32_t source);
<b>Function descriptions</b>	disable RTC tamper
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	specify which tamper source to be disabled
<i>RTC_TAMPER0</i>	RTC tamper0
<i>RTC_TAMPER1</i>	RTC tamper1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable RTC tamper */
rtc_tamper_disable(RTC_TAMPER0);
```

## rtc\_software\_bkp\_reset

The description of rtc\_software\_bkp\_reset is shown as below:

**Table 3-762. Function rtc\_software\_bkp\_reset**

<b>Function name</b>	rtc_software_bkp_reset
<b>Function prototype</b>	void rtc_software_bkp_reset(void)
<b>Function descriptions</b>	reset the RTC_BKP registers by software
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset the RTC_BKP registers by software */
rtc_software_bkp_reset ();
```



## rtc\_tamper\_without\_bkp\_seset

The description of rtc\_tamper\_without\_bkp\_seset is shown as below:

**Table 3-763. Function rtc\_tamper\_without\_bkp\_seset**

<b>Function name</b>	rtc_tamper_without_bkp_seset
<b>Function prototype</b>	void rtc_tamper_without_bkp_seset(uint32_t ne_source)
<b>Function descriptions</b>	tamperx event does not erase the RTC_BKP registers
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ne_source</b>	specify which tamper source will not trigger the RTC_BKP registers reset
<i>RTC_TAMPXNOER_NONE</i>	both tamper0 and tamper1 event will trigger RTC_BKP registers reset
<i>RTC_TAMPXNOER_TP0</i>	tamper0 event will not trigger RTC_BKP registers reset
<i>RTC_TAMPXNOER_TP1</i>	tamper1 event will not trigger RTC_BKP registers reset
<i>RTC_TAMPXNOER_TP0_TP1</i>	neither tamper0 nor tamper1 event will trigger RTC_BKP registers reset
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set tamper0 will not trigger RTC_BKP registers */
rtc_tamper_without_bkp_seset(RTC_TAMPXNOER_TP0);
```

## rtc\_output\_pin\_select

The description of rtc\_output\_pin\_select is shown as below:

**Table 3-764. Function rtc\_output\_pin\_select**

<b>Function name</b>	rtc_output_pin_select
<b>Function prototype</b>	void rtc_output_pin_select(uint32_t pad);
<b>Function descriptions</b>	select the RTC output pin pad
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pad</b>	specify the rtc output pad is PC15 or not
<i>RTC_OUT_PC15</i>	the rtc output pad is PC15

<i>RTC_OUT_PA3_PA8</i>	the rtc output pad is PA3 or PA8 according to the AFIO configuration
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select the rtc output pad is PC15 */
```

```
rtc_output_pin_select(RTC_OUT_PC15);
```

### rtc\_alarm\_output\_config

The description of `rtc_alarm_output_config` is shown as below:

**Table 3-765. Function `rtc_alarm_output_config`**

<b>Function name</b>	<code>rtc_alarm_output_config</code>
<b>Function prototype</b>	<code>void rtc_alarm_output_config(uint32_t source, uint32_t mode);</code>
<b>Function descriptions</b>	configure rtc alarm output source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	specify signal to output
<i>RTC_ALARM0_HIGH</i>	when the alarm0 flag is set, the output pin is high
<i>RTC_ALARM0_LOW</i>	when the alarm0 flag is set, the output pin is low
<i>RTC_ALARM1_HIGH</i>	when the alarm1 flag is set, the output pin is high
<i>RTC_ALARM1_LOW</i>	when the alarm1 flag is set, the output pin is low
<i>RTC_WAKEUP_HIGH</i>	when the wakeup flag is set, the output pin is high
<i>RTC_WAKEUP_LOW</i>	when the wakeup flag is set, the output pin is low
<b>Input parameter{in}</b>	
<b>mode</b>	specify the output pin mode when output alarm signal or auto wakeup signal
<i>RTC_ALARM_OUTPUT_OD</i>	open drain mode
<i>RTC_ALARM_OUTPUT_PP</i>	push pull mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure rtc alternate output source */
```

```
rtc_alarm_output_config(RTC_ALARM0_LOW, RTC_ALARM_OUTPUT_PP);
```

## rtc\_calibration\_config

The description of rtc\_calibration\_config is shown as below:

**Table 3-766. Function rtc\_calibration\_config**

<b>Function name</b>	rtc_calibration_config
<b>Function prototype</b>	void rtc_calibration_output_config(uint32_t source);
<b>Function descriptions</b>	configure rtc calibration output source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	specify signal to output
<i>RTC_CALIBRATION_512HZ</i>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 512Hz signal
<i>RTC_CALIBRATION_1HZ</i>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 1Hz signal
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure rtc calibration output source */
```

```
rtc_calibration_output_config(RTC_CALIBRATION_1HZ);
```

## rtc\_hour\_adjust

The description of rtc\_hour\_adjust is shown as below:

**Table 3-767. Function rtc\_hour\_adjust**

<b>Function name</b>	rtc_hour_adjust
<b>Function prototype</b>	void rtc_hour_adjust(uint32_t operation);
<b>Function descriptions</b>	adjust the daylight saving time by adding or subtracting one hour from the current time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>operation</b>	hour adjustment operation
<i>RTC_CTL_A1H</i>	add one hour
<i>RTC_CTL_S1H</i>	subtract one hour
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* adjust the daylight saving time by adding one hour from the current time */
```

```
rtc_hour_adjust(RTC_CTL_A1H);
```

### rtc\_second\_adjust

The description of rtc\_second\_adjust is shown as below:

**Table 3-768. Function rtc\_second\_adjust**

<b>Function name</b>	rtc_second_adjust
<b>Function prototype</b>	ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);
<b>Function descriptions</b>	adjust RTC second or subsecond value of current time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>add</b>	add 1s to current time or not
<i>RTC_SHIFT_ADD1S_R ESET</i>	no effect
<i>RTC_SHIFT_ADD1S_S ET</i>	add 1s to current time
<b>Input parameter{in}</b>	
<b>minus</b>	number of subsecond to minus from current time(0x0 - 0x7FFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* adjust RTC second or subsecond value of current time */
```

```
ErrStatus error_status = rtc_second_adjust(RTC_SHIFT_ADD1S_SET, 0);
```

### rtc\_bypass\_shadow\_enable

The description of rtc\_bypass\_shadow\_enable is shown as below:

**Table 3-769. Function rtc\_bypass\_shadow\_enable**

<b>Function name</b>	rtc_bypass_shadow_enable
<b>Function prototype</b>	void rtc_bypass_shadow_enable(void);
<b>Function descriptions</b>	enable RTC bypass shadow registers function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable RTC bypass shadow registers function */
```

```
rtc_bypass_shadow_enable();
```

## rtc\_bypass\_shadow\_disable

The description of rtc\_bypass\_shadow\_disable is shown as below:

**Table 3-770. Function rtc\_bypass\_shadow\_disable**

Function name	rtc_bypass_shadow_disable
Function prototype	void rtc_bypass_shadow_disable (void);
Function descriptions	disable RTC bypass shadow registers function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RTC bypass shadow registers function */
```

```
rtc_bypass_shadow_disable ();
```

## rtc\_refclock\_detection\_enable

The description of rtc\_refclock\_detection\_enable is shown as below:

**Table 3-771. Function rtc\_refclock\_detection\_enable**

Function name	rtc_refclock_detection_enable
Function prototype	ErrStatus rtc_refclock_detection_enable(void);
Function descriptions	enable RTC reference clock detection function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* enable RTC reference clock detection function*/
```

```
ErrStatus error_status = rtc_refclock_detection_enable();
```

### rtc\_refclock\_detection\_disable

The description of rtc\_refclock\_detection\_disable is shown as below:

**Table 3-772. Function rtc\_refclock\_detection\_disable**

Function name	rtc_refclock_detection_disable
Function prototype	ErrStatus rtc_refclock_detection_disable(void);
Function descriptions	disable RTC reference clock detection function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* disable RTC reference clock detection function*/
```

```
ErrStatus error_status = rtc_refclock_detection_disable ();
```

### rtc\_wakeup\_enable

The description of rtc\_refclock\_detection\_disable is shown as below:

**Table 3-773. Function rtc\_wakeup\_enable**

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

Example:

```
/* enable RTC auto wakeup function */
```

```
rtc_wakeup_enable();
```

### rtc\_wakeup\_disable

The description of rtc\_wakeup\_disable is shown as below:

**Table 3-774. Function rtc\_wakeup\_disable**

<b>Function name</b>	rtc_wakeup_disable
<b>Function prototype</b>	ErrStatus rtc_wakeup_disable(void);
<b>Function descriptions</b>	disable RTC auto wakeup function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* disable RTC auto wakeup function */
```

```
ErrStatus error_status = rtc_wakeup_disable ();
```

### rtc\_wakeup\_clock\_set

The description of rtc\_wakeup\_clock\_set is shown as below:

**Table 3-775. Function rtc\_wakeup\_clock\_set**

<b>Function name</b>	rtc_wakeup_clock_set
<b>Function prototype</b>	ErrStatus rtc_wakeup_clock_set(uint8_t wakeup_clock);
<b>Function descriptions</b>	set RTC auto wakeup timer clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_clock</b>	wakeup timer clock is RTC clock divided factor
WAKEUP_RTCK_DIV 16	RTC auto wakeup timer clock is RTC clock divided by 16
WAKEUP_RTCK_DIV 8	RTC auto wakeup timer clock is RTC clock divided by 8
WAKEUP_RTCK_DIV 4	RTC auto wakeup timer clock is RTC clock divided by 4

<i>WAKEUP_RTCK_DIV</i> 2	RTC auto wakeup timer clock is RTC clock divided by 2
<i>WAKEUP_CKSPRE</i>	RTC auto wakeup timer clock is ckspre
<i>WAKEUP_CKSPRE_2</i> <i>EXP16</i>	RTC auto wakeup timer clock is ckspre and wakeup timer add 2exp16
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* disable RTC auto wakeup function */
```

```
ErrStatus error_status = rtc_wakeup_clock_set(WAKEUP_RTCK_DIV8);
```

### rtc\_wakeup\_timer\_set

The description of rtc\_wakeup\_timer\_set is shown as below:

**Table 3-776. Function rtc\_wakeup\_timer\_set**

<b>Function name</b>	rtc_wakeup_timer_set
<b>Function prototype</b>	ErrStatus rtc_wakeup_timer_set(uint16_t wakeup_timer);
<b>Function descriptions</b>	set wakeup timer value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_timer</b>	wakeup timer value
<i>uint16_t</i>	0x0000-0xffff
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* set wakeup timer value */
```

```
ErrStatus error_status = rtc_wakeup_timer_set(0XFFEE);
```

### rtc\_wakeup\_timer\_get

The description of rtc\_wakeup\_timer\_set is shown as below:

**Table 3-777. Function rtc\_wakeup\_timer\_get**

<b>Function name</b>	rtc_wakeup_timer_get
<b>Function prototype</b>	uint16_t rtc_wakeup_timer_get(void);
<b>Function descriptions</b>	set wakeup timer value



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	0-0xFFFF

Example:

```
/* get wakeup timer value */
```

```
uint32_t wakeup_time = rtc_wakeup_timer();
```

### rtc\_smooth\_calibration\_config

The description of rtc\_smooth\_calibration\_config is shown as below:

**Table 3-778. rtc\_smooth\_calibration\_config**

<b>Function name</b>	rtc_smooth_calibration_config
<b>Function prototype</b>	ErrStatus rtc_smooth_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);
<b>Function descriptions</b>	configure RTC smooth calibration
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>window</b>	select calibration window
<i>RTC_CALIBRATION_WINDOW_32S</i>	2exp20 RTCCLK cycles, 32s if RTCCLK = 32768 Hz
<i>RTC_CALIBRATION_WINDOW_16S</i>	2exp19 RTCCLK cycles, 16s if RTCCLK = 32768 Hz
<i>RTC_CALIBRATION_WINDOW_8S</i>	2exp18 RTCCLK cycles, 8s if RTCCLK = 32768 Hz
<b>Input parameter{in}</b>	
<b>plus</b>	add RTC clock or not
<i>RTC_CALIBRATION_PLUS_SET</i>	add one RTC clock every 2048 rtc clock
<i>RTC_CALIBRATION_PLUS_RESET</i>	no effect
<b>Input parameter{in}</b>	
<b>minus</b>	the RTC clock to minus during the calibration window(0x0 - 0x1FF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

<b>ErrStatus</b>	ERROR or SUCCESS
------------------	------------------

Example:

```
/* configure RTC smooth calibration */
```

```
ErrStatus error_status;
```

```
error_status = rtc_smooth_calibration_config(RTC_CALIBRATION_WINDOW_32S, RTC_CALIBRATION_PLUS_SET, 0x10);
```

## rtc\_coarse\_calibration\_enable

The description of rtc\_coarse\_calibration\_enable is shown as below:

**Table 3-779. rtc\_coarse\_calibration\_enable**

<b>Function name</b>	rtc_coarse_calibration_enable
<b>Function prototype</b>	ErrStatus rtc_coarse_calibration_enable(void);
<b>Function descriptions</b>	enable RTC coarse calibration
<b>Precondition</b>	-
<b>The called functions</b>	rtc_init_mode_enter/rtc_init_mode_exit
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* enable RTC coarse calibration */
```

```
rtc_coarse_calibration_enable ();
```

## rtc\_coarse\_calibration\_disable

The description of rtc\_coarse\_calibration\_disable is shown as below:

**Table 3-780. rtc\_coarse\_calibration\_disable**

<b>Function name</b>	rtc_coarse_calibration_disable
<b>Function prototype</b>	ErrStatus rtc_coarse_calibration_disable(void);
<b>Function descriptions</b>	disable RTC coarse calibration
<b>Precondition</b>	-
<b>The called functions</b>	rtc_init_mode_enter/rtc_init_mode_exit
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* rtc_coarse_calibration_disable */
```

```
ErrStatus error_status = rtc_coarse_calibration_disable ();
```

### rtc\_coarse\_calibration\_config

The description of rtc\_coarse\_calibration\_config is shown as below:

**Table 3-781. rtc\_coarse\_calibration\_config**

Function name	rtc_coarse_calibration_config
Function prototype	ErrStatus rtc_coarse_calibration_config(uint8_t direction, uint8_t step);
Function descriptions	config coarse calibration direction and step
Precondition	-
The called functions	rtc_init_mode_enter/rtc_init_mode_exit
Input parameter{in}	
direction	coarse calibration direction
CALIB_INCREASE	Increase calendar update frequency
CALIB_DECREASE	Decrease calendar update frequency
Input parameter{in}	
step	coarse calibration step
0x00-0x1F	COSD=0: 0x00: +0PPM 0x01: +4PPM(approximate value) 0x02: +8PPM (approximate value) ..... 0x1F: +126PPM (approximate value) COSD=1: 0x00: -0PPM 0x01: -2PPM(approximate value) 0x02: -4PPM (approximate value) ..... 0x1F: -63PPM (approximate value)
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* config coarse calibration direction and step */
```

```
ErrStatus error_status = rtc_coarse_calibration_config (INCREASE, 0x01);
```

## rtc\_pri\_pro\_enable

The description of rtc\_pri\_pro\_enable is shown as below:

**Table 3-782. rtc\_pri\_pro\_enable**

<b>Function name</b>	rtc_pri_pro_enable
<b>Function prototype</b>	void rtc_pri_pro_enable(uint32_t sub_area);
<b>Function descriptions</b>	enable RTC privilege protection mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sub_area</b>	specify which sub area of the RTC module to be protected by the privilege mode
<i>RTC_PPM_CTL_ALRM0PRIP</i>	Alarm 0 privilege protection
<i>RTC_PPM_CTL_ALRM1PRIP</i>	Alarm 1 privilege protection
<i>RTC_PPM_CTL_WUTPRIP</i>	Wakeup timer privilege protection
<i>RTC_PPM_CTL_TSPRIP</i>	Timestamp privilege protection
<i>RTC_PPM_CTL_TAMPPRIP</i>	Tamper privilege protection (excluding backup registers)
<i>RTC_PPM_CTL_CALCPRIP</i>	Shift register, daylight saving, calibration and reference clock privilege protection
<i>RTC_PPM_CTL_INITPRIP</i>	Initialization privilege protection
<i>RTC_PPM_CTL_RTCPRIP</i>	RTC privilege protection
<i>RTC_PPM_CTL_BKPRIP</i>	backup registers zone a privilege protection
<i>RTC_PPM_CTL_BKPWPRIP</i>	backup registers zone b privilege protection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable RTC alarm 0 privilege protection mode */
rtc_pri_pro_enable(RTC_PPM_CTL_ALRM0PRIP);
```

## rtc\_pri\_pro\_disable

The description of rtc\_pri\_pro\_disable is shown as below:

**Table 3-783. rtc\_pri\_pro\_enable**

<b>Function name</b>	rtc_pri_pro_disable
<b>Function prototype</b>	void rtc_pri_pro_disable(uint32_t sub_area);
<b>Function descriptions</b>	disable RTC privilege protection mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sub_area</b>	specify which sub area of the RTC module to be disabled protecte by the privilege mode
<i>RTC_PPM_CTL_ALRM0PRIP</i>	Alarm 0 privilege protection
<i>RTC_PPM_CTL_ALRM1PRIP</i>	Alarm 1 privilege protection
<i>RTC_PPM_CTL_WUTPRIP</i>	Wakeup timer privilege protection
<i>RTC_PPM_CTL_TSPRIP</i>	Timestamp privilege protection
<i>RTC_PPM_CTL_TAMPPRIP</i>	Tamper privilege protection (excluding backup registers)
<i>RTC_PPM_CTL_CALCPRIP</i>	Shift register, daylight saving, calibration and reference clock privilege protection
<i>RTC_PPM_CTL_INITPRIP</i>	Initialization privilege protection
<i>RTC_PPM_CTL_RTCPRIP</i>	RTC privilege protection
<i>RTC_PPM_CTL_BKPRWPRIP</i>	backup registers zone a privilege protection
<i>RTC_PPM_CTL_BKPWPRIP</i>	backup registers zone b privilege protection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable RTC alarm 0 privilege protection mode */
```

```
rtc_pri_pro_disable(RTC_PPM_CTL_ALRM0PRIP);
```

## rtc\_sec\_pro\_enable

The description of rtc\_sec\_pro\_enable is shown as below:

**Table 3-784. rtc\_sec\_pro\_enable**

<b>Function name</b>	rtc_sec_pro_enable
<b>Function prototype</b>	void rtc_sec_pro_enable(uint32_t sub_area);
<b>Function descriptions</b>	enable RTC secure protection mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sub_area</b>	specify which sub area of the RTC module to be protected by the secure mode
<i>RTC_SPM_CTL_ALRM0SECP</i>	Alarm 0 secure protection
<i>RTC_SPM_CTL_ALRM1SECP</i>	Alarm 1 secure protection
<i>RTC_SPM_CTL_WUTSECP</i>	Wakeup timer secure protection
<i>RTC_SPM_CTL_TSSECP</i>	Timestamp secure protection
<i>RTC_SPM_CTL_TAMPSECP</i>	Tamper secure protection (excluding backup registers)
<i>RTC_SPM_CTL_CALCSECP</i>	Shift register, daylight saving, calibration and reference clock secure protection
<i>RTC_SPM_CTL_INITSECP</i>	Initialization secure protection
<i>RTC_SPM_CTL_RTCSSECP</i>	RTC global secure protection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable RTC alarm 0 secure protection mode */
rtc_sec_pro_enable (RTC_SPM_CTL_ALRM0SECP);
```

## rtc\_sec\_pro\_disable

The description of rtc\_sec\_pro\_disable is shown as below:

**Table 3-785. rtc\_sec\_pro\_disable**

<b>Function name</b>	rtc_sec_pro_disable
----------------------	---------------------

<b>Function prototype</b>	void rtc_sec_pro_disable(uint32_t sub_area);
<b>Function descriptions</b>	disable RTC secure protection mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sub_area</b>	specify which sub area of the RTC module to be disabled protecte by the secure mode
<i>RTC_SPM_CTL_ALRM0SECP</i>	Alarm 0 secure protection
<i>RTC_SPM_CTL_ALRM1SECP</i>	Alarm 1 secure protection
<i>RTC_SPM_CTL_WUTSECP</i>	Wakeup timer secure protection
<i>RTC_SPM_CTL_TSSECP</i>	Timestamp secure protection
<i>RTC_SPM_CTL_TAMPSECP</i>	Tamper secure protection (excluding backup registers)
<i>RTC_SPM_CTL_CALCSECP</i>	Shift register, daylight saving, calibration and reference clock secure protection
<i>RTC_SPM_CTL_INITSECP</i>	Initialization secure protection
<i>RTC_SPM_CTL_RTCSSECP</i>	RTC global secure protection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*disable RTC alarm 0 secure protection mode */
```

```
rtc_sec_pro_disable (RTC_SPM_CTL_ALRM0SECP);
```

## rtc\_bkp\_zone\_a\_sec\_pro\_set

The description of rtc\_bkp\_zone\_a\_sec\_pro\_set is shown as below:

**Table 3-786. rtc\_bkp\_zone\_a\_sec\_pro\_set**

<b>Function name</b>	rtc_bkp_zone_a_sec_pro_set
<b>Function prototype</b>	void rtc_bkp_zone_a_sec_pro_set(uint32_t zone_a_tail);
<b>Function descriptions</b>	set the RTC_BKP secure protection zone_a tail value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>zonea_tail</b>	RTC_BKP registers secure protection zonea
<i>RTC_BKP_PRO_ZONE</i> <i>A_TAIL_NONE</i>	there is no RTC_BKP registers secure protection zonea
<i>RTC_BKP_PRO_ZONE</i> <i>A_TAIL_x</i>	RTC_BKP registers secure protection zonea is from RTC_BKP0 to RTC_BKPx, which can be read and written only when the APB is in secure mode (x = 0,1,2,...19)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the RTC_BKP secure protection zonea tail value */
```

```
rtc_bkp_zoneb_sec_pro_set (RTC_BKP_PRO_ZONEA_TAIL_3);
```

### rtc\_bkp\_zoneb\_sec\_pro\_set

The description of rtc\_bkp\_zoneb\_sec\_pro\_set is shown as below:

**Table 3-787. rtc\_bkp\_zoneb\_sec\_pro\_set**

<b>Function name</b>	rtc_bkp_zoneb_sec_pro_set
<b>Function prototype</b>	ErrStatus rtc_bkp_zoneb_sec_pro_set(uint32_t zoneb_tail);
<b>Function descriptions</b>	set the RTC_BKP secure protection zoneb tail value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>zoneb_tail</b>	RTC_BKP registers secure protection zoneb
<i>RTC_BKP_PRO_ZONE</i> <i>B_TAIL_NONE</i>	there is no RTC_BKP registers secure protection zoneb
<i>RTC_BKP_PRO_ZONE</i> <i>B_TAIL_x</i>	RTC_BKP registers secure protection zonea is from RTC_BKP0 to RTC_BKPx, which can be read and written only when the APB is in secure mode (x = 0,1,2,...19)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
ErrStatus	ERROR or SUCCESS

Example:

```
/* set the RTC_BKP secure protection zoneb tail value */
```

```
ErrStatus error_states = rtc_bkp_zoneb_sec_pro_set(RTC_BKP_PRO_ZONEB_TAIL_3);
```



## rtc\_bkp\_zoneb\_sec\_pro\_check

The description of rtc\_bkp\_zoneb\_sec\_pro\_check is shown as below:

**Table 3-788. rtc\_bkp\_zoneb\_sec\_pro\_check**

<b>Function name</b>	rtc_bkp_zoneb_sec_pro_check
<b>Function prototype</b>	ErrStatus rtc_bkp_zoneb_sec_pro_check(void);
<b>Function descriptions</b>	check the RTC_BKP secure protection zoneb is valid or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
ErrStatus	ERROR or SUCCESS

Example:

```
/* check the RTC_BKP secure protection zoneb is valid or not */
```

```
ErrStatus error_states = rtc_bkp_zoneb_sec_pro_check ();
```

## rtc\_flag\_get

The description of rtc\_flag\_get is shown as below:

**Table 3-789. Function rtc\_flag\_get**

<b>Function name</b>	rtc_flag_get
<b>Function prototype</b>	FlagStatus rtc_flag_get(uint32_t flag);
<b>Function descriptions</b>	check specified flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag to check
<i>RTC_FLAG_SCP</i>	smooth calibration pending flag
<i>RTC_FLAG_TP1</i>	RTC tamper 1 detected flag
<i>RTC_FLAG_TP0</i>	RTC tamper 0 detected flag
<i>RTC_FLAG_TSOVR</i>	time-stamp overflow event flag
<i>RTC_FLAG_TS</i>	time-stamp event flag
<i>RTC_FLAG_ALARM0</i>	alarm0 event flag
<i>RTC_FLAG_ALARM1</i>	alarm1 event flag
<i>RTC_FLAG_WT</i>	wakeup timer occurs flag
<i>RTC_FLAG_INIT</i>	initialization state flag

<i>RTC_FLAG_RSYN</i>	register synchronization flag
<i>RTC_FLAG_YCM</i>	year configuration mark status flag
<i>RTC_FLAG_SOP</i>	shift function operation pending flag
<i>RTC_FLAG_ALARM0W</i>	alarm0 configuration can be written flag
<i>RTC_FLAG_ALARM1W</i>	alarm1 configuration can be written flag
<i>RTC_FLAG_WTW</i>	wakeup timer can be written flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
FlagStatus	SET or RESET

Example:

```
/* check time-stamp event flag */
```

```
FlagStatus = rtc_flag_get(RTC_FLAG_TS);
```

## rtc\_flag\_clear

The description of `rtc_flag_clear` is shown as below:

**Table 3-790. Function `rtc_flag_clear`**

<b>Function name</b>	<code>rtc_flag_clear</code>
<b>Function prototype</b>	<code>void rtc_flag_clear(uint32_t flag);</code>
<b>Function descriptions</b>	clear specified flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag to clear
<i>RTC_FLAG_TP1</i>	RTC tamper 1 detected flag
<i>RTC_FLAG_TP0</i>	RTC tamper 0 detected flag
<i>RTC_FLAG_TSOVR</i>	time-stamp overflow event flag
<i>RTC_FLAG_TS</i>	time-stamp event flag
<i>RTC_FLAG_WT</i>	wakeup timer occurs flag
<i>RTC_FLAG_ALARM0</i>	alarm0 occurs flag
<i>RTC_FLAG_ALARM1</i>	alarm1 occurs flag
<i>RTC_FLAG_RSYN</i>	register synchronization flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear time-stamp event flag */
```

```
rtc_flag_clear (RTC_FLAG_TS);
```

## rtc\_interrupt\_enable

The description of rtc\_interrupt\_enable is shown as below:

**Table 3-791. Function rtc\_interrupt\_enable**

<b>Function name</b>	rtc_interrupt_enable
<b>Function prototype</b>	void rtc_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable specified RTC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which interrupt source to be enabled
<i>RTC_INT_TIMESTAMP</i>	timestamp interrupt
<i>RTC_INT_ALARM0</i>	alarm0 interrupt
<i>RTC_INT_ALARM1</i>	alarm1 interrupt
<i>RTC_INT_WAKEUP</i>	wakeup timer interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable specified RTC interrupt*/
rtc_interrupt_enable(RTC_INT_ALARM0);
```

## rtc\_interrupt\_disable

The description of rtc\_interrupt\_disable is shown as below:

**Table 3-792. Function rtc\_interrupt\_disable**

<b>Function name</b>	rtc_interrupt_disable
<b>Function prototype</b>	void rtc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disble specified RTC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which RTC interrupt to disable
<i>RTC_INT_TIMESTAMP</i>	timestamp interrupt
<i>RTC_INT_ALARM0</i>	alarm0 interrupt
<i>RTC_INT_ALARM1</i>	alarm1 interrupt
<i>RTC_INT_WAKEUP</i>	wakeup timer interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* disable specified RTC interrupt */  
  
rtc_interrupt_disable(RTC_INT_ALARM0);
```

## rtc\_nsec\_interrupt\_flag\_get

The description of rtc\_nsec\_interrupt\_flag\_get is shown as below:

**Table 3-793. Function rtc\_nsec\_interrupt\_flag\_get**

<b>Function name</b>	rtc_nsec_interrupt_flag_get
<b>Function prototype</b>	FlagStatus rtc_nsec_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	get specified non-secure interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	specify which flag to clear
<i>RTC_NSMI_STAT_ALRMONSMF</i>	alarm0 non-secure interrupt masked flag
<i>RTC_NSMI_STAT_ALRM1NSMF</i>	alarm1 non-secure interrupt masked flag
<i>RTC_NSMI_STAT_WTNSMF</i>	wakeup timer non-secure interrupt masked flag
<i>RTC_NSMI_STAT_TSNSMF</i>	time-stamp non-secure interrupt masked flag
<i>RTC_NSMI_STAT_TSOVRNSMF</i>	time-stamp overflow non-secure interrupt masked flag
<i>RTC_NSMI_STAT_TPONSMF</i>	RTC_TAMP0 non-secure interrupt masked flag
<i>RTC_NSMI_STAT_TP1NSMF</i>	RTC_TAMP1 non-secure interrupt masked flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get alarm0 non-secure interrupt flag */  
  
rtc_nsec_interrupt_flag_get(RTC_NSMI_STAT_ALRMONSMF);
```

## rtc\_nsec\_interrupt\_flag\_clear

The description of `rtc_nsec_interrupt_flag_clear` is shown as below:

**Table 3-794. Function `rtc_nsec_interrupt_flag_clear`**

<b>Function name</b>	<code>rtc_nsec_interrupt_flag_clear</code>
<b>Function prototype</b>	<code>void rtc_nsec_interrupt_flag_clear(uint32_t int_flag)</code>
<b>Function descriptions</b>	clear specified non-secure interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	specify which flag to clear
<code>RTC_NSMI_STAT_ALRM0NSMF</code>	alarm0 non-secure interrupt masked flag
<code>RTC_NSMI_STAT_ALRM1NSMF</code>	alarm1 non-secure interrupt masked flag
<code>RTC_NSMI_STAT_WTNSMF</code>	wakeup timer non-secure interrupt masked flag
<code>RTC_NSMI_STAT_TSNSMF</code>	time-stamp non-secure interrupt masked flag
<code>RTC_NSMI_STAT_TSOVRNSMF</code>	time-stamp overflow non-secure interrupt masked flag
<code>RTC_NSMI_STAT_TP0NSMF</code>	RTC_TAMP0 non-secure interrupt masked flag
<code>RTC_NSMI_STAT_TP1NSMF</code>	RTC_TAMP1 non-secure interrupt masked flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear alarm0 non-secure interrupt flag */
rtc_nsec_interrupt_flag_clear(RTC_NSMI_STAT_ALRM0NSMF);
```

## rtc\_sec\_interrupt\_flag\_get

The description of `rtc_sec_interrupt_flag_get` is shown as below:

**Table 3-795. Function `rtc_sec_interrupt_flag_get`**

<b>Function name</b>	<code>rtc_sec_interrupt_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus rtc_sec_interrupt_flag_get(uint32_t int_flag);</code>
<b>Function descriptions</b>	get specified secure interrupt flag
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	specify which flag to get
<i>RTC_SMI_STAT_ALR M0SMF</i>	alarm0 secure interrupt masked flag
<i>RTC_SMI_STAT_ALR M1SMF</i>	alarm1 secure interrupt masked flag
<i>RTC_SMI_STAT_WTS MF</i>	wakeup timer secure interrupt masked flag
<i>RTC_SMI_STAT_TSS MF</i>	time-stamp secure interrupt masked flag
<i>RTC_SMI_STAT_TSO VRSMF</i>	time-stamp overflow secure interrupt masked flag
<i>RTC_SMI_STAT_TP0S MF</i>	RTC_TAMP0 secure interrupt masked flag
<i>RTC_SMI_STAT_TP1S MF</i>	RTC_TAMP1 secure interrupt masked flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get alarm0 secure interrupt flag */
```

```
rtc_sec_interrupt_flag_get(RTC_SMI_STAT_ALRM0SMF);
```

## rtc\_sec\_interrupt\_flag\_clear

The description of rtc\_sec\_interrupt\_flag\_clear is shown as below:

**Table 3-796. Function rtc\_nsec\_interrupt\_flag\_clear**

<b>Function name</b>	rtc_sec_interrupt_flag_clear
<b>Function prototype</b>	void rtc_sec_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear specified secure interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	specify which flag to clear
<i>RTC_SMI_STAT_ALR M0SMF</i>	alarm0 secure interrupt masked flag
<i>RTC_SMI_STAT_ALR M1SMF</i>	alarm1 secure interrupt masked flag
<i>RTC_SMI_STAT_WTS</i>	wakeup timer secure interrupt masked flag

<i>MF</i>	
<i>RTC_SMI_STAT_TSS</i> <i>MF</i>	time-stamp secure interrupt masked flag
<i>RTC_SMI_STAT_TSO</i> <i>VRSMF</i>	time-stamp overflow secure interrupt masked flag
<i>RTC_SMI_STAT_TP0S</i> <i>MF</i>	RTC_TAMP0 secure interrupt masked flag
<i>RTC_SMI_STAT_TP1S</i> <i>MF</i>	RTC_TAMP1 secure interrupt masked flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear alarm0 secure interrupt flag */
```

```
rtc_sec_interrupt_flag_clear(RTC_SMI_STAT_ALRM0SMF);
```

## 3.24. 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 [3.24.1](#), the SDIO firmware functions are introduced in chapter [3.24.2](#).

### 3.24.1. Descriptions of Peripheral registers

SDIO registers are listed in the table shown as below:

**Table 3-797. 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

Registers	Descriptions
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

### 3.24.2. Descriptions of Peripheral functions

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

**Table 3-798. SDIO firmware function**

Function name	Function description
sdio_deinit	deinitialize the SDIO
sdio_clock_config	configure the SDIO clock
sdio_hardware_clock_enable	enable hardware clock control
sdio_hardware_clock_disable	disable hardware clock control
sdio_bus_mode_set	set different SDIO card bus mode
sdio_power_state_set	set the SDIO power state
sdio_power_state_get	get the SDIO power state
sdio_clock_enable	enable SDIO_CLK clock output
sdio_clock_disable	disable SDIO_CLK clock output
sdio_command_response_config	configure the command and response
sdio_wait_type_set	set the command state machine wait type
sdio_csm_enable	enable the CSM(command state machine)
sdio_csm_disable	disable the CSM(command state machine)
sdio_command_index_get	get the last response command index
sdio_response_get	get the response for the last received command
sdio_data_config	configure the data timeout, data length and data block size
sdio_data_transfer_config	configure the data transfer mode and direction
sdio_dsm_enable	enable the DSM(data state machine) for data transfer
sdio_dsm_disable	disable the DSM(data state machine)
sdio_data_write	write data(one word) to the transmit FIFO
sdio_data_read	read data(one word) from the receive FIFO
sdio_data_counter_get	get the number of remaining data bytes to be transferred to card
sdio_fifo_counter_get	get the number of words remaining to be written or read from FIFO
sdio_dma_enable	enable the DMA request for SDIO
sdio_dma_disable	disable the DMA request for SDIO
sdio_flag_get	get the flags state of SDIO
sdio_flag_clear	clear the pending flags of SDIO



Function name	Function description
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 3-799. Function sdio\_deinit**

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

Example:

```
/* deinitialize the SDIO */
```

```
sdio_deinit();
```

## sdio\_clock\_config

The description of sdio\_clock\_config is shown as below:

**Table 3-800. Function sdio\_clock\_config**

<b>Function name</b>	sdio_clock_config
<b>Function prototype</b>	void sdio_clock_config(uint32_t clock_edge, uint32_t clock_bypass, uint32_t clock_powersave, uint16_t clock_division);
<b>Function descriptions</b>	configure the SDIO clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>clock_edge</b>	SDIO_CLK clock edge
SDIO_SDIOLCKEDGE_RISING	select the rising edge of the SDIOCLK to generate SDIO_CLK
SDIO_SDIOLCKEDGE_FALLING	select the falling edge of the SDIOCLK to generate SDIO_CLK
<b>Input parameter{in}</b>	
<b>clock_bypass</b>	clock bypass
SDIO_CLOCKBYPASS_ENABLE	clock bypass
SDIO_CLOCKBYPASS_DISABLE	no bypass
<b>Input parameter{in}</b>	
<b>clock_powersave</b>	SDIO_CLK clock dynamic switch on/off for power saving
SDIO_CLOCKPWRSAVE_ENABLE	SDIO_CLK closed when bus is idle
SDIO_CLOCKPWRSAVE_DISABLE	SDIO_CLK clock is always on
<b>Input parameter{in}</b>	
<b>clock_division</b>	clock division, less than 512
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the SDIO clock */
```

```
sdio_clock_config(SDIO_SDIOLCKEDGE_RISING, SDIO_CLOCKBYPASS_DISABLE,
SDIO_CLOCKPWRSAVE_DISABLE, SD_CLK_DIV_TRANS);
```

## sdio\_hardware\_clock\_enable

The description of sdio\_hardware\_clock\_enable is shown as below:

**Table 3-801. Function sdio\_hardware\_clock\_enable**

<b>Function name</b>	sdio_hardware_clock_enable
<b>Function prototype</b>	void sdio_hardware_clock_enable(void);
<b>Function descriptions</b>	enable hardware clock control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-802. Function sdio\_hardware\_clock\_disable**

<b>Function name</b>	sdio_hardware_clock_disable
<b>Function prototype</b>	void sdio_hardware_clock_disable(void);
<b>Function descriptions</b>	disable hardware clock control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-803. Function sdio\_bus\_mode\_set**

<b>Function name</b>	sdio_bus_mode_set
<b>Function prototype</b>	void sdio_bus_mode_set(uint32_t bus_mode);
<b>Function descriptions</b>	set different SDIO card bus mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>bus_mode</b>	SDIO card bus mode
SDIO_BUSMODE_1BIT T	1-bit SDIO card bus mode
SDIO_BUSMODE_4BIT T	4-bit SDIO card bus mode
SDIO_BUSMODE_8BIT T	8-bit SDIO card bus mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-804. Function sdio\_power\_state\_set**

<b>Function name</b>	sdio_power_state_set
<b>Function prototype</b>	void sdio_power_state_set(uint32_t power_state);
<b>Function descriptions</b>	set the SDIO power state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>power_state</b>	SDIO power state
SDIO_POWER_ON	SDIO power on
SDIO_POWER_OFF	SDIO power off
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

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 3-805. Function sdio\_power\_state\_get**

<b>Function name</b>	sdio_power_state_get
<b>Function prototype</b>	uint32_t sdio_power_state_get(void);
<b>Function descriptions</b>	get the SDIO power state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-806. Function sdio\_clock\_enable**

<b>Function name</b>	sdio_clock_enable
<b>Function prototype</b>	void sdio_clock_enable(void);
<b>Function descriptions</b>	enable SDIO_CLK clock output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* enable SDIO_CLK clock output */
```

```
sdio_clock_enable();
```

### sdio\_clock\_disable

The description of sdio\_clock\_disable is shown as below:

**Table 3-807. Function sdio\_clock\_disable**

<b>Function name</b>	sdio_clock_disable
<b>Function prototype</b>	void sdio_clock_disable(void);
<b>Function descriptions</b>	disable SDIO_CLK clock output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-808. Function sdio\_command\_response\_config**

<b>Function name</b>	sdio_command_response_config
<b>Function prototype</b>	void sdio_command_response_config(uint32_t cmd_index, uint32_t cmd_argument, uint32_t response_type);
<b>Function descriptions</b>	configure the command and response
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmd_index</b>	command index, refer to the related specifications
<b>Input parameter{in}</b>	
<b>cmd_argument</b>	command argument, refer to the related specifications
<b>Input parameter{in}</b>	
<b>response_type</b>	response type

<i>SDIO_RESPONSETYPE_NO</i>	no response
<i>SDIO_RESPONSETYPE_SHORT</i>	short response
<i>SDIO_RESPONSETYPE_LONG</i>	long response
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-809. Function sdio\_wait\_type\_set**

<b>Function name</b>	sdio_wait_type_set
<b>Function prototype</b>	void sdio_wait_type_set(uint32_t wait_type);
<b>Function descriptions</b>	set the command state machine wait type
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wait_type</b>	wait type
<i>SDIO_WAITTYPE_NO</i>	not wait interrupt
<i>SDIO_WAITTYPE_INTERRUPT</i>	wait interrupt
<i>SDIO_WAITTYPE_DATAEND</i>	wait the end of data transfer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-810. Function sdio\_csm\_enable**

<b>Function name</b>	sdio_csm_enable
<b>Function prototype</b>	void sdio_csm_enable(void);
<b>Function descriptions</b>	enable the CSM(command state machine)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the CSM(command state machine) */
sdio_csm_enable();
```

## sdio\_csm\_disable

The description of sdio\_csm\_disable is shown as below:

**Table 3-811. Function sdio\_csm\_disable**

<b>Function name</b>	sdio_csm_disable
<b>Function prototype</b>	void sdio_csm_disable(void);
<b>Function descriptions</b>	disable the CSM(command state machine)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-812. Function sdio\_command\_index\_get**

<b>Function name</b>	sdio_command_index_get
<b>Function prototype</b>	uint8_t sdio_command_index_get(void);
<b>Function descriptions</b>	get the last response command index
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	last response command index

Example:

```
/* get SDIO command index */
uint8_t sdio_commond_value;
sdio_commond_value = sdio_command_index_get();
```

## sdio\_response\_get

The description of sdio\_response\_get is shown as below:

**Table 3-813. Function sdio\_response\_get**

<b>Function name</b>	sdio_response_get
<b>Function prototype</b>	uint32_t sdio_response_get(uint32_t responsex);
<b>Function descriptions</b>	get the response for the last received command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	response for the last received command

Example:

```
/* store the CID0 numbers */
```

```
uint32_t sdio_cid[4] = {0, 0, 0, 0};
```

```
sdio_cid[0] = sdio_response_get(SDIO_RESPONSE0);
```

### sdio\_data\_config

The description of sdio\_data\_config is shown as below:

**Table 3-814. Function sdio\_data\_config**

<b>Function name</b>	sdio_data_config
<b>Function prototype</b>	void sdio_data_config(uint32_t data_timeout, uint32_t data_length, uint32_t data_blocksize);
<b>Function descriptions</b>	configure the data timeout, data length and data block size
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data_timeout</b>	data timeout period in card bus clock periods
<b>Input parameter{in}</b>	
<b>data_length</b>	number of data bytes to be transferred
<b>Input parameter{in}</b>	
<b>data_blocksize</b>	size of data block for block transfer
SDIO_DATABLOCKSIZE_1BYTE	block size = 1 byte
SDIO_DATABLOCKSIZE_2BYTES	block size = 2 bytes
SDIO_DATABLOCKSIZE_4BYTES	block size = 4 bytes
SDIO_DATABLOCKSIZE_8BYTES	block size = 8 bytes
SDIO_DATABLOCKSIZE_16BYTES	block size = 16 bytes
SDIO_DATABLOCKSIZE_32BYTES	block size = 32 bytes
SDIO_DATABLOCKSIZE_64BYTES	block size = 64 bytes
SDIO_DATABLOCKSIZE_128BYTES	block size = 128 bytes
SDIO_DATABLOCKSIZE_256BYTES	block size = 256 bytes
SDIO_DATABLOCKSIZE_512BYTES	block size = 512 bytes
SDIO_DATABLOCKSIZE_1024BYTES	block size = 1024 bytes

<i>SDIO_DATABLOCKSIZE_2048BYTES</i>	block size = 2048 bytes
<i>SDIO_DATABLOCKSIZE_4096BYTES</i>	block size = 4096 bytes
<i>SDIO_DATABLOCKSIZE_8192BYTES</i>	block size = 8192 bytes
<i>SDIO_DATABLOCKSIZE_16384BYTES</i>	block size = 16384 bytes
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-815. Function sdio\_data\_transfer\_config**

<b>Function name</b>	sdio_data_transfer_config
<b>Function prototype</b>	void sdio_data_transfer_config(uint32_t transfer_mode, uint32_t transfer_direction);
<b>Function descriptions</b>	configure the data transfer mode and direction
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>transfer_mode</b>	mode of data transfer
<i>SDIO_TRANSMODE_BLOCK</i>	block transfer
<i>SDIO_TRANSMODE_STREAM</i>	stream transfer or SDIO multibyte transfer
<b>Input parameter{in}</b>	
<b>transfer_direction</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* configure SDIO data transmisson */
```

```
sdio_data_transfer_config(SDIO_TRANSDIRECTION_TOSDIO,
SDIO_TRANSMODE_BLOCK);
```

### sdio\_dsm\_enable

The description of sdio\_dsm\_enable is shown as below:

**Table 3-816. Function sdio\_dsm\_enable**

<b>Function name</b>	sdio_dsm_enable
<b>Function prototype</b>	void sdio_dsm_enable(void);
<b>Function descriptions</b>	enable the DSM(data state machine) for data transfer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the DSM(data state machine) */
```

```
sdio_dsm_enable();
```

### sdio\_dsm\_disable

The description of sdio\_dsm\_disable is shown as below:

**Table 3-817. Function sdio\_dsm\_disable**

<b>Function name</b>	sdio_dsm_disable
<b>Function prototype</b>	void sdio_dsm_disable(void);
<b>Function descriptions</b>	disable the DSM(data state machine)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the DSM(data state machine) */
```

```
sdio_dsm_disable();
```

## sdio\_data\_write

The description of sdio\_data\_write is shown as below:

**Table 3-818. Function sdio\_data\_write**

<b>Function name</b>	sdio_data_write
<b>Function prototype</b>	void sdio_data_write(uint32_t data);
<b>Function descriptions</b>	write data(one word) to the transmit FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data</b>	32-bit data write to card
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-819. Function sdio\_data\_read**

<b>Function name</b>	sdio_data_read
<b>Function prototype</b>	uint32_t sdio_data_read(void);
<b>Function descriptions</b>	read data(one word) from the receive FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	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 3-820. Function sdio\_data\_counter\_get**

<b>Function name</b>	sdio_data_counter_get
<b>Function prototype</b>	uint32_t sdio_data_counter_get(void);
<b>Function descriptions</b>	get the number of remaining data bytes to be transferred to card
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-821. Function sdio\_data\_counter\_get**

<b>Function name</b>	sdio_fifo_counter_get
<b>Function prototype</b>	uint32_t sdio_fifo_counter_get(void);
<b>Function descriptions</b>	get the number of words remaining to be written or read from FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-822. Function sdio\_dma\_enable**

<b>Function name</b>	sdio_dma_enable
<b>Function prototype</b>	void sdio_dma_enable(void);
<b>Function descriptions</b>	enable the DMA request for SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the SDIO DMA */
```

```
sdio_dma_enable();
```

## sdio\_dma\_disable

The description of sdio\_dma\_disable is shown as below:

**Table 3-823. Function sdio\_dma\_disable**

<b>Function name</b>	sdio_dma_disable
<b>Function prototype</b>	void sdio_dma_disable(void);
<b>Function descriptions</b>	disable the DMA request for SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the SDIO DMA */
```

```
sdio_dma_disable();
```

## sdio\_flag\_get

The description of sdio\_flag\_get is shown as below:

**Table 3-824. Function sdio\_flag\_get**

<b>Function name</b>	sdio_flag_get
<b>Function prototype</b>	FlagStatus sdio_flag_get(uint32_t flag);
<b>Function descriptions</b>	get the flags state of SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flags state of SDIO
<i>SDIO_FLAG_CCRCE</i> <i>R</i>	command response received (CRC check failed) flag
<i>SDIO_FLAG_DTCRCE</i> <i>RR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_FLAG_CMDTMO</i> <i>UT</i>	command response timeout flag
<i>SDIO_FLAG_DTTMOU</i> <i>T</i>	data timeout flag
<i>SDIO_FLAG_TXURE</i>	transmit FIFO underrun error occurs flag
<i>SDIO_FLAG_RXORE</i>	received FIFO overrun error occurs flag
<i>SDIO_FLAG_CMDREC</i> <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_DTBKE</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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-825. Function `sdio_flag_clear`**

<b>Function name</b>	<code>sdio_flag_clear</code>
<b>Function prototype</b>	<code>void sdio_flag_clear(uint32_t flag);</code>
<b>Function descriptions</b>	clear the pending flags of SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flags state of SDIO
<i>SDIO_FLAG_CCR CER</i> <i>R</i>	command response received (CRC check failed) flag
<i>SDIO_FLAG_DTCRCE</i> <i>RR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_FLAG_CMDTMO</i> <i>UT</i>	command response timeout flag
<i>SDIO_FLAG_DTTMOU</i> <i>T</i>	data timeout flag
<i>SDIO_FLAG_TXURE</i>	transmit FIFO underrun error occurs flag
<i>SDIO_FLAG_RXORE</i>	received FIFO overrun error occurs flag
<i>SDIO_FLAG_CMDREC</i> <i>V</i>	command response received (CRC check passed) flag
<i>SDIO_FLAG_CMDSEN</i> <i>D</i>	command sent (no response required) flag
<i>SDIO_FLAG_DTEND</i>	data end (data counter, <code>SDIO_DATA_CNT</code> , is zero) flag
<i>SDIO_FLAG_STBITE</i>	start bit error in the bus flag

<i>SDIO_FLAG_DTBLKE</i> <i>ND</i>	data block sent/received (CRC check passed) flag
<i>SDIO_FLAG_SDIOINT</i>	SD I/O interrupt received flag
<i>SDIO_FLAG_ATAEND</i>	CE-ATA command completion signal received (only for CMD61) flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-826. Function sdio\_interrupt\_enable**

<b>Function name</b>	sdio_interrupt_enable
<b>Function prototype</b>	void sdio_interrupt_enable(uint32_t int_flag);
<b>Function descriptions</b>	enable the SDIO interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flags state of SDIO
<i>SDIO_INT_CCRRCERR</i>	SDIO CCRRCERR interrupt
<i>SDIO_INT_DTCRCERR</i> <i>R</i>	SDIO DTCRCERR interrupt
<i>SDIO_INT_CMDTMOUT</i> <i>T</i>	SDIO CMDTMOUT interrupt
<i>SDIO_INT_DTTMOUT</i>	SDIO DTTMOUT interrupt
<i>SDIO_INT_TXURE</i>	SDIO TXURE interrupt
<i>SDIO_INT_RXORE</i>	SDIO_INT_RXORE
<i>SDIO_INT_CMDRECV</i>	SDIO CMDRECV interrupt
<i>SDIO_INT_CMDSEND</i>	SDIO CMDSEND interrupt
<i>SDIO_INT_DTEND</i>	SDIO DTEND interrupt
<i>SDIO_INT_STBITE</i>	SDIO STBITE interrupt
<i>SDIO_INT_DTBLKEND</i>	SDIO DTBLKEND interrupt
<i>SDIO_INT_CMDRUN</i>	SDIO CMDRUN interrupt
<i>SDIO_INT_TXRUN</i>	SDIO TXRUN interrupt
<i>SDIO_INT_RXRUN</i>	SDIO RXRUN interrupt
<i>SDIO_INT_TFH</i>	SDIO TFH interrupt
<i>SDIO_INT_RFH</i>	SDIO RFH interrupt

<i>SDIO_INT_TFF</i>	SDIO TFF interrupt
<i>SDIO_INT_RFF</i>	SDIO RFF interrupt
<i>SDIO_INT_TFE</i>	SDIO TFE interrupt
<i>SDIO_INT_RFE</i>	SDIO RFE interrupt
<i>SDIO_INT_TXDTVAL</i>	SDIO TXDTVAL interrupt
<i>SDIO_INT_RXDTVAL</i>	SDIO RXDTVAL interrupt
<i>SDIO_INT_SDIOINT</i>	SDIO SDIOINT interrupt
<i>SDIO_INT_ATAEND</i>	SDIO ATAEND interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the SDIO corresponding interrupts */
```

```
sdio_interrupt_enable(SDIO_INT_CCRRCERR | 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 3-827. Function sdio\_interrupt\_disable**

<b>Function name</b>	sdio_interrupt_disable
<b>Function prototype</b>	void sdio_interrupt_disable(uint32_t int_flag);
<b>Function descriptions</b>	disable the SDIO interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flags state of SDIO
<i>SDIO_INT_CCRRCERR</i>	SDIO CCRRCERR interrupt
<i>SDIO_INT_DTCRCERR</i> <i>R</i>	SDIO DTCRCERR interrupt
<i>SDIO_INT_CMDTMOUT</i> <i>T</i>	SDIO CMDTMOUT interrupt
<i>SDIO_INT_DTTMOUT</i>	SDIO DTTMOUT interrupt
<i>SDIO_INT_TXURE</i>	SDIO TXURE interrupt
<i>SDIO_INT_RXORE</i>	SDIO_INT_RXORE
<i>SDIO_INT_CMDRECV</i>	SDIO CMDRECV interrupt
<i>SDIO_INT_CMDSND</i>	SDIO CMDSND 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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-828. Function `sdio_interrupt_flag_get`**

<b>Function name</b>	<code>sdio_interrupt_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus sdio_interrupt_flag_get(uint32_t int_flag);</code>
<b>Function descriptions</b>	get the interrupt flags state of SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	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</i>	SDIO TXURE interrupt flag

<i>RE</i>	
<i>SDIO_INT_FLAG_RXORE</i>	SDIO_INT_RXORE flag
<i>SDIO_INT_FLAG_CMDRECV</i>	SDIO CMDRECV interrupt flag
<i>SDIO_INT_FLAG_CMDSEND</i>	SDIO CMDSEND interrupt flag
<i>SDIO_INT_FLAG_DTEND</i>	SDIO DTEND interrupt flag
<i>SDIO_INT_FLAG_STBITE</i>	SDIO STBITE interrupt flag
<i>SDIO_INT_FLAG_DTBLKEND</i>	SDIO DTBLKEND interrupt flag
<i>SDIO_INT_FLAG_CMDRUN</i>	SDIO CMDRUN interrupt flag
<i>SDIO_INT_FLAG_TXRUN</i>	SDIO TXRUN interrupt flag
<i>SDIO_INT_FLAG_RXRUN</i>	SDIO RXRUN interrupt flag
<i>SDIO_INT_FLAG_TFH</i>	SDIO TFH interrupt flag
<i>SDIO_INT_FLAG_RFH</i>	SDIO RFH interrupt flag
<i>SDIO_INT_FLAG_TFF</i>	SDIO TFF interrupt flag
<i>SDIO_INT_FLAG_RFF</i>	SDIO RFF interrupt flag
<i>SDIO_INT_FLAG_TFE</i>	SDIO TFE interrupt flag
<i>SDIO_INT_FLAG_RFE</i>	SDIO RFE interrupt flag
<i>SDIO_INT_FLAG_TXDTVAL</i>	SDIO TXDTVAL interrupt flag
<i>SDIO_INT_FLAG_RXDTVAL</i>	SDIO RXDTVAL interrupt flag
<i>SDIO_INT_FLAG_SDIOINT</i>	SDIO SDIOINT interrupt flag
<i>SDIO_INT_FLAG_ATAEND</i>	SDIO ATAEND interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-829. Function sdio\_interrupt\_flag\_clear**

<b>Function name</b>	sdio_interrupt_flag_clear
<b>Function prototype</b>	void sdio_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear the interrupt pending flags of SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flags state of SDIO
SDIO_INT_FLAG_CCR CERR	command response received (CRC check failed) flag
SDIO_INT_FLAG_DTC RCERR	data block sent/received (CRC check failed) flag
SDIO_INT_FLAG_CMD TMOUT	command response timeout flag
SDIO_INT_FLAG_DTT MOUT	data timeout flag
SDIO_INT_FLAG_TXU RE	transmit FIFO underrun error occurs flag
SDIO_INT_FLAG_RXO RE	received FIFO overrun error occurs flag
SDIO_INT_FLAG_CMD RECV	command response received (CRC check passed) flag
SDIO_INT_FLAG_CMD SEND	command sent (no response required) flag
SDIO_INT_FLAG_DTE ND	data end (data counter, SDIO_DATACNT, is zero) flag
SDIO_INT_FLAG_STBI TE	start bit error in the bus flag
SDIO_INT_FLAG_DTB LKEND	data block sent/received (CRC check passed) flag
SDIO_INT_FLAG_SDI OINT	SD I/O interrupt received flag
SDIO_INT_FLAG_ATA END	CE-ATA command completion signal received (only for CMD61) flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-830. Function sdio\_readwait\_enable**

<b>Function name</b>	sdio_readwait_enable
<b>Function prototype</b>	void sdio_readwait_enable(void);
<b>Function descriptions</b>	enable the read wait mode(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-831. Function sdio\_readwait\_disable**

<b>Function name</b>	sdio_readwait_disable
<b>Function prototype</b>	void sdio_readwait_disable(void);
<b>Function descriptions</b>	disable the read wait mode(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-832. Function sdio\_stop\_readwait\_enable**

<b>Function name</b>	sdio_stop_readwait_enable
<b>Function prototype</b>	void sdio_stop_readwait_enable(void);
<b>Function descriptions</b>	enable the function that stop the read wait process(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-833. Function sdio\_stop\_readwait\_disable**

<b>Function name</b>	sdio_stop_readwait_disable
<b>Function prototype</b>	void sdio_stop_readwait_disable(void);
<b>Function descriptions</b>	disable the function that stop the read wait process(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-834. Function sdio\_readwait\_type\_set**

<b>Function name</b>	sdio_readwait_type_set
<b>Function prototype</b>	void sdio_readwait_type_set(uint32_t readwait_type);
<b>Function descriptions</b>	set the read wait type(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>readwait_type</b>	SD I/O read wait type
SDIO_READWAITTYPE_CLK	read wait control by stopping SDIO_CLK
SDIO_READWAITTYPE_DAT2	read wait control using SDIO_DAT[2]
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-835. Function sdio\_operation\_enable**

<b>Function name</b>	sdio_operation_enable
<b>Function prototype</b>	void sdio_operation_enable(void);
<b>Function descriptions</b>	enable the SD I/O mode specific operation(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-836. Function sdio\_operation\_disable**

<b>Function name</b>	sdio_operation_disable
<b>Function prototype</b>	void sdio_operation_disable(void);
<b>Function descriptions</b>	disable the SD I/O mode specific operation(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-837. Function sdio\_suspend\_enable**

<b>Function name</b>	sdio_suspend_enable
<b>Function prototype</b>	void sdio_suspend_enable(void);
<b>Function descriptions</b>	enable the SD I/O suspend operation(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-838. Function sdio\_suspend\_disable**

<b>Function name</b>	sdio_suspend_disable
<b>Function prototype</b>	void sdio_suspend_disable(void);
<b>Function descriptions</b>	disable the SD I/O suspend operation(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-839. Function sdio\_ceata\_command\_enable**

<b>Function name</b>	sdio_ceata_command_enable
<b>Function prototype</b>	void sdio_ceata_command_enable(void);
<b>Function descriptions</b>	enable the CE-ATA command(CE-ATA only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-840. Function sdio\_ceata\_command\_disable**

<b>Function name</b>	sdio_ceata_command_disable
<b>Function prototype</b>	void sdio_ceata_command_disable(void);
<b>Function descriptions</b>	disable the CE-ATA command(CE-ATA only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-841. Function sdio\_ceata\_interrupt\_enable**

<b>Function name</b>	sdio_ceata_interrupt_enable
<b>Function prototype</b>	void sdio_ceata_interrupt_enable(void);
<b>Function descriptions</b>	enable the CE-ATA interrupt(CE-ATA only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-842. Function sdio\_ceata\_interrupt\_disable**

<b>Function name</b>	sdio_ceata_interrupt_disable
<b>Function prototype</b>	void sdio_ceata_interrupt_disable(void);
<b>Function descriptions</b>	disable the CE-ATA interrupt(CE-ATA only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-843. Function sdio\_ceata\_command\_completion\_enable**

<b>Function name</b>	sdio_ceata_command_completion_enable
<b>Function prototype</b>	void sdio_ceata_command_completion_enable(void);
<b>Function descriptions</b>	enable the CE-ATA command completion signal(CE-ATA only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-844. Function sdio\_ceata\_command\_completion\_disable**

<b>Function name</b>	sdio_ceata_command_completion_disable
<b>Function prototype</b>	void sdio_ceata_command_completion_disable(void);
<b>Function descriptions</b>	disable the CE-ATA command completion signal(CE-ATA only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the CE-ATA command completion signal(CE-ATA only) */
sdio_ceata_command_completion_disable();
```

## 3.25. 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 [3.25.1](#), the SPI/I2S firmware functions are introduced in chapter [3.25.2](#).

### 3.25.1. Descriptions of Peripheral registers

SPI/I2S registers are listed in the table shown as below:

**Table 3-845. SPI/I2S Registers**

Registers	Descriptions
SPI_CTL0	SPI control register 0
SPI_CTL1	SPI control register 1
SPI_STAT	SPI status register
SPI_DATA	SPI data register
SPI_CRCPOLY	SPI CRC polynomial register
SPI_RCRC	SPI receive CRC register
SPI_TCRC	SPI transmit CRC register
SPI_I2SCTL	SPI/I2S control register
SPI_I2SPSC	SPI/I2S clock prescaler register
SPI_QCTL	Quad-SPI mode control register

### 3.25.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

**Table 3-846. 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_ckin_psc_config	configure I2S input clock 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
i2s_full_duplex_mode_config	configure i2s full duplex mode
spi_i2s_format_error_clear	clear TI mode format error flag status
spi_crc_polynomial_set	set SPI CRC polynomial
spi_crc_polynomial_get	get SPI CRC polynomial
spi_crc_on	turn on SPI CRC function
spi_crc_off	turn off SPI CRC function
spi_crc_next	SPI next data is CRC value
spi_crc_get	get SPI CRC send value or receive value
spi_crc_error_clear	clear SPI CRC error flag status
spi_ti_mode_enable	enable SPI TI mode
spi_ti_mode_disable	disable SPI TI mode
spi_quad_enable	enable quad wire SPI
spi_quad_disable	disable quad wire SPI
spi_quad_write_enable	enable quad wire SPI write
spi_quad_read_enable	enable quad wire SPI read
spi_i2s_flag_get	get SPI and I2S flag status
spi_i2s_interrupt_enable	enable SPI and I2S interrupt

Function name	Function description
spi_i2s_interrupt_disable	disable SPI and I2S interrupt
spi_i2s_interrupt_flag_get	get SPI and I2S interrupt status

### Structure spi\_parameter\_struct

**Table 3-847. Structure 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_FRAME_SIZE_16BIT, SPI_FRAME_SIZE_8BIT)
nss	SPI NSS control by hardware or software (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	SPI big endian or little endian (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	SPI clock phase and polarity (SPI_CLOCK_PL_LOW_PH_1EDGE, SPI_CLOCK_PL_HIGH_PH_1EDGE, SPI_CLOCK_PL_LOW_PH_2EDGE, SPI_CLOCK_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 3-848. Function spi\_i2s\_deinit**

<b>Function name</b>	spi_i2s_deinit
<b>Function prototype</b>	void spi_i2s_deinit(uint32_t spi_periph);
<b>Function descriptions</b>	reset SPI and I2S peripheral
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI/I2S peripheral
<i>SPIx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:



```
/* reset SPI0 */
```

```
spi_i2s_deinit(SPI0);
```

## spi\_struct\_para\_init

The description of spi\_struct\_para\_init is shown as below:

**Table 3-849. Function spi\_struct\_para\_init**

<b>Function name</b>	spi_struct_para_init
<b>Function prototype</b>	void spi_struct_para_init(spi_parameter_struct* spi_struct);
<b>Function descriptions</b>	initialize the parameters of SPI struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>*spi_struct</b>	a spi_parameter_struct address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-850. Function spi\_init**

<b>Function name</b>	spi_init
<b>Function prototype</b>	void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
<b>Function descriptions</b>	initialize SPI peripheral parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>spi_struct</b>	SPI parameter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-847. Structure spi_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* initialize SPI0 */
```

```
spi_parameter_struct spi_init_struct;
```

```
spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
```

```
spi_init_struct.device_mode     = SPI_MASTER;
```

```
spi_init_struct.frame_size     = SPI_FRAME_SIZE_8BIT;
```

```
spi_init_struct.clock_polarity_phase = SPI_CLOCK_POLARITY_HIGH_PHASE_2EDGE;
```

```
spi_init_struct.nss            = SPI_NSS_SOFT;
```

```
spi_init_struct.prescale       = SPI_PRESCALE_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 3-851. Function spi\_enable**

<b>Function name</b>	spi_enable
<b>Function prototype</b>	void spi_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 */
```

```
spi_enable(SPI0);
```

## spi\_disable

The description of spi\_disable is shown as below:

**Table 3-852. Function spi\_disable**

<b>Function name</b>	spi_disable
<b>Function prototype</b>	void spi_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 */
spi_disable(SPI0);
```

## i2s\_init

The description of i2s\_init is shown as below:

**Table 3-853. Function i2s\_init**

<b>Function name</b>	i2s_init
<b>Function prototype</b>	void i2s_init(uint32_t spi_periph, uint32_t i2s_mode, uint32_t i2s_standard, uint32_t i2s_ckpl);
<b>Function descriptions</b>	initialize I2S peripheral parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<i>SPIx</i>	x=1
<b>Input parameter{in}</b>	
<b>i2s_mode</b>	I2S operation mode
<i>I2S_MODE_SLAVETX</i>	I2S slave transmit mode
<i>I2S_MODE_SLAVERX</i>	I2S slave receive mode
<i>I2S_MODE_MASTERTX</i>	I2S master transmit mode
<i>I2S_MODE_MASTERRX</i>	I2S master receive mode
<b>Input parameter{in}</b>	
<b>i2s_standard</b>	I2S standard
<i>I2S_STD_PHILIPS</i>	I2S philips standard

<i>I2S_STD_MSB</i>	I2S MSB standard
<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCMSHORT</i>	I2S PCM short standard
<i>I2S_STD_PCMLONG</i>	I2S PCM long standard
<b>Input parameter{in}</b>	
<b>i2s_ckpl</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize I2S1 */
```

```
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

### i2s\_psc\_config

The description of i2s\_psc\_config is shown as below:

**Table 3-854. Function i2s\_psc\_config**

<b>Function name</b>	i2s_psc_config
<b>Function prototype</b>	void i2s_psc_config(uint32_t spi_periph, uint32_t i2s_audiosample, uint32_t i2s_frameformat, uint32_t i2s_mckout);
<b>Function descriptions</b>	configure I2S prescaler
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<i>SPIx</i>	x=1
<b>Input parameter{in}</b>	
<b>i2s_audiosample</b>	I2S audio sample rate
<i>I2S_AUDIOSAMPLE_8K</i>	audio sample rate is 8KHz
<i>I2S_AUDIOSAMPLE_11K</i>	audio sample rate is 11KHz
<i>I2S_AUDIOSAMPLE_16K</i>	audio sample rate is 16KHz
<i>I2S_AUDIOSAMPLE_22K</i>	audio sample rate is 22KHz
<i>I2S_AUDIOSAMPLE_32K</i>	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
<b>Input parameter{in}</b>	
<b>i2s_frameformat</b>	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
<b>Input parameter{in}</b>	
<b>i2s_mckout</b>	I2S master clock output
<i>I2S_MCKOUT_ENABL</i> <i>E</i>	I2S master clock output enable
<i>I2S_MCKOUT_DISABL</i> <i>E</i>	I2S master clock output disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2S1 prescaler */
```

```
i2s_psc_config(SPI1, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,  
I2S_MCKOUT_DISABLE);
```

### **i2s\_ckin\_psc\_config**

The description of i2s\_ckin\_psc\_config is shown as below:

**Table 3-855. Function i2s1\_ckin\_psc\_config**

<b>Function name</b>	i2s_ckin_psc_config
<b>Function prototype</b>	void i2s_ckin_psc_config(uint32_t input_clock, uint32_t i2s_audiosample, uint32_t i2s_frameformat, uint32_t i2s_mckout);
<b>Function descriptions</b>	configure I2S input clock prescaler

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>input_clock</b>	I2S1 input clock
<b>Input parameter{in}</b>	
<b>i2s_audiosample</b>	I2S audio sample rate
<i>I2S_AUDIOSAMPLE_8K</i>	audio sample rate is 8KHz
<i>I2S_AUDIOSAMPLE_11K</i>	audio sample rate is 11KHz
<i>I2S_AUDIOSAMPLE_16K</i>	audio sample rate is 16KHz
<i>I2S_AUDIOSAMPLE_22K</i>	audio sample rate is 22KHz
<i>I2S_AUDIOSAMPLE_32K</i>	audio sample rate is 32KHz
<i>I2S_AUDIOSAMPLE_44K</i>	audio sample rate is 44KHz
<i>I2S_AUDIOSAMPLE_48K</i>	audio sample rate is 48KHz
<i>I2S_AUDIOSAMPLE_96K</i>	audio sample rate is 96KHz
<i>I2S_AUDIOSAMPLE_192K</i>	audio sample rate is 192KHz
<b>Input parameter{in}</b>	
<b>i2s_frameformat</b>	I2S data length and channel length
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S data length is 16 bit and channel length is 16 bit
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S data length is 16 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S data length is 24 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S data length is 32 bit and channel length is 32 bit
<b>Input parameter{in}</b>	
<b>i2s_mckout</b>	I2S master clock output
<i>I2S_MCKOUT_ENABLER</i>	I2S master clock output enable
<i>I2S_MCKOUT_DISABLE</i>	I2S master clock output disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* configure I2S1 input clock prescaler */
```

```
#define I2S1_INPUT_CK 50000000
```

```
i2s_psc_config(I2S1_INPUT_CK, I2S_AUDIOSAMPLE_8K, I2S_FRAMEFORMAT_DT16B
_CH16B, I2S_MCKOUT_DISABLE);
```

## i2s\_enable

The description of i2s\_enable is shown as below:

**Table 3-856. Function i2s\_enable**

<b>Function name</b>	i2s_enable
<b>Function prototype</b>	void i2s_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable I2S
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<i>SPI1</i>	I2S1 peripheral
<i>I2S1_ADD</i>	I2S1_ADD peripheral
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2S1*/
```

```
i2s_enable(SPI1);
```

## i2s\_disable

The description of i2s\_disable is shown as below:

**Table 3-857. Function i2s\_disable**

<b>Function name</b>	i2s_disable
<b>Function prototype</b>	void i2s_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable I2S
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral

<i>SPI1</i>	I2S1 peripheral
<i>I2S1_ADD</i>	I2S1_ADD peripheral
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2S1*/
i2s_disable(SPI1);
```

### **spi\_nss\_output\_enable**

The description of spi\_nss\_output\_enable is shown as below:

**Table 3-858. Function spi\_nss\_output\_enable**

<b>Function name</b>	spi_nss_output_enable
<b>Function prototype</b>	void spi_nss_output_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI NSS output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-859. Function spi\_nss\_output\_disable**

<b>Function name</b>	spi_nss_output_disable
<b>Function prototype</b>	void spi_nss_output_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable SPI NSS output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	



<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-860. Function spi\_nss\_internal\_high**

<b>Function name</b>	spi_nss_internal_high
<b>Function prototype</b>	void spi_nss_internal_high(uint32_t spi_periph);
<b>Function descriptions</b>	SPI NSS pin high level in software mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-861. Function spi\_nss\_internal\_low**

<b>Function name</b>	spi_nss_internal_low
<b>Function prototype</b>	void spi_nss_internal_low(uint32_t spi_periph);
<b>Function descriptions</b>	SPI NSS pin low level in software mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-862. Function spi\_dma\_enable**

<b>Function name</b>	spi_dma_enable
<b>Function prototype</b>	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
<b>Function descriptions</b>	enable SPI DMA function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>dma</b>	SPI DMA mode
<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA
<i>SPI_DMA_RECEIVE</i>	SPI receive data use DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-863. Function spi\_dma\_disable**

<b>Function name</b>	spi_dma_disable
<b>Function prototype</b>	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);

<b>Function descriptions</b>	disable SPI DMA function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>dma</b>	SPI DMA mode
<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA
<i>SPI_DMA_RECEIVE</i>	SPI receive data use DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-864. Function spi\_i2s\_data\_frame\_format\_config**

<b>Function name</b>	spi_i2s_data_frame_format_config
<b>Function prototype</b>	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
<b>Function descriptions</b>	configure SPI/I2S data frame format
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>frame_format</b>	SPI frame size
<i>SPI_FRAME_SIZE_16BIT</i>	SPI frame size is 16 bits
<i>SPI_FRAME_SIZE_8BIT</i>	SPI frame size is 8 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SPI1/I2S1 data frame format size is 16 bits */
```

```
spi_i2s_data_frame_format_config(SPI1, SPI_FRAME_SIZE_16BIT);
```

## spi\_i2s\_data\_transmit

The description of spi\_i2s\_data\_transmit is shown as below:

**Table 3-865. Function spi\_i2s\_data\_transmit**

<b>Function name</b>	spi_i2s_data_transmit
<b>Function prototype</b>	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
<b>Function descriptions</b>	SPI transmit data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>data</b>	16-bit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-866. Function spi\_i2s\_data\_receive**

<b>Function name</b>	spi_i2s_data_receive
<b>Function prototype</b>	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
<b>Function descriptions</b>	SPI receive data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-

Return value	
<b>uint16_t</b>	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 3-867. Function spi\_bidirectional\_transfer\_config**

<b>Function name</b>	spi_bidirectional_transfer_config
<b>Function prototype</b>	void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);
<b>Function descriptions</b>	configure SPI bidirectional transfer direction
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>transfer_direction</b>	SPI transfer direction
<i>SPI_BIDIRECTIONAL_TRANSMIT</i>	SPI work in transmit-only mode
<i>SPI_BIDIRECTIONAL_RECEIVE</i>	SPI work in receive-only mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 works in transmit-only mode */
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

## i2s\_full\_duplex\_mode\_config

The description of i2s\_full\_duplex\_mode\_config is shown as below:

**Table 3-868. Function i2s\_init**

<b>Function name</b>	i2s_full_duplex_mode_config
----------------------	-----------------------------

<b>Function prototype</b>	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);
<b>Function descriptions</b>	configure i2s full duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2s_add_periph</b>	I2S_ADD peripheral
<i>I2Sx_ADD</i>	x=1
<b>Input parameter{in}</b>	
<b>i2s_mode</b>	I2S operation mode
<i>I2S_MODE_SLAVETX</i>	I2S slave transmit mode
<i>I2S_MODE_SLAVRX</i>	I2S slave receive mode
<i>I2S_MODE_MASTERT</i> X	I2S master transmit mode
<i>I2S_MODE_MASTERR</i> X	I2S master receive mode
<b>Input parameter{in}</b>	
<b>i2s_standard</b>	I2S standard
<i>I2S_STD_PHILIPS</i>	I2S philips standard
<i>I2S_STD_MSB</i>	I2S MSB standard
<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCMSHORT</i>	I2S PCM short standard
<i>I2S_STD_PCMLONG</i>	I2S PCM long standard
<b>Input parameter{in}</b>	
<b>i2s_ckpl</b>	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
<b>Input parameter{in}</b>	
<b>i2s_frameformat</b>	I2S data length and channel length
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S data length is 16 bit and channel length is 16 bit
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S data length is 16 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S data length is 24 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S data length is 32 bit and channel length is 32 bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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\_i2s\_format\_error\_clear

The description of spi\_i2s\_format\_error\_clear is shown as below:

**Table 3-869. Function spi\_i2s\_format\_error\_clear**

<b>Function name</b>	spi_i2s_format_error_clear
<b>Function prototype</b>	void spi_i2s_format_error_clear(uint32_t spi_periph, uint32_t flag);
<b>Function descriptions</b>	clear TI mode format error flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>flag</b>	SPI/I2S frame format error flag
<i>SPI_FLAG_FERR</i>	SPI TI mode frame format error
<i>I2S_FLAG_FERR</i>	I2S frame format error
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear SPI0 TI mode format error flag */
```

```
spi_i2s_format_error_clear(SPI0, SPI_FLAG_FERR);
```

## spi\_crc\_polynomial\_set

The description of spi\_crc\_polynomial\_set is shown as below:

**Table 3-870. Function spi\_crc\_polynomial\_set**

<b>Function name</b>	spi_crc_polynomial_set
<b>Function prototype</b>	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);
<b>Function descriptions</b>	set SPI CRC polynomial
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral

<i>SPIx</i>	<i>x</i> =0,1
<b>Input parameter{in}</b>	
<b>crc_poly</b>	CRC polynomial value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-871. Function spi\_crc\_polynomial\_get**

<b>Function name</b>	spi_crc_polynomial_get
<b>Function prototype</b>	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
<b>Function descriptions</b>	get SPI CRC polynomial
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	<i>x</i> =0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	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 3-872. Function spi\_crc\_on**

<b>Function name</b>	spi_crc_on
<b>Function prototype</b>	void spi_crc_on(uint32_t spi_periph);
<b>Function descriptions</b>	turn on SPI CRC function



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SP/x</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn on SPI0 CRC function */
```

```
spi_crc_on(SPI0);
```

## spi\_crc\_off

The description of spi\_crc\_off is shown as below:

**Table 3-873. Function spi\_crc\_off**

<b>Function name</b>	spi_crc_off
<b>Function prototype</b>	void spi_crc_off(uint32_t spi_periph);
<b>Function descriptions</b>	turn off SPI CRC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SP/x</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn off SPI0 CRC function */
```

```
spi_crc_off(SPI0);
```

## spi\_crc\_next

The description of spi\_crc\_next is shown as below:

**Table 3-874. Function spi\_crc\_next**

<b>Function name</b>	spi_crc_next
<b>Function prototype</b>	void spi_crc_next(uint32_t spi_periph);
<b>Function descriptions</b>	SPI next data is CRC value

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-875. Function spi\_crc\_get**

<b>Function name</b>	spi_crc_get
<b>Function prototype</b>	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc);
<b>Function descriptions</b>	get SPI CRC send value or receive value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>crc</b>	SPI crc value
<b>SPI_CRC_TX</b>	get transmit crc value
<b>SPI_CRC_RX</b>	get receive crc value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	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 3-876. Function spi\_crc\_error\_clear**

<b>Function name</b>	spi_crc_error_clear
<b>Function prototype</b>	void spi_crc_error_clear(uint32_t spi_periph);
<b>Function descriptions</b>	clear SPI CRC error flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-877. Function spi\_ti\_mode\_enable**

<b>Function name</b>	spi_ti_mode_enable
<b>Function prototype</b>	void spi_ti_mode_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI TI mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-878. Function spi\_ti\_mode\_disable**

<b>Function name</b>	spi_ti_mode_disable
<b>Function prototype</b>	void spi_ti_mode_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable SPI TI mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

## **spi\_quad\_enable**

The description of spi\_quad\_enable is shown as below:

**Table 3-879. Function spi\_quad\_enable**

<b>Function name</b>	spi_quad_enable
<b>Function prototype</b>	void spi_quad_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable quad wire SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 quad wire mode */
spi_quad_enable(SPI0);
```

## spi\_quad\_disable

The description of spi\_quad\_disable is shown as below:

**Table 3-880. Function spi\_quad\_disable**

<b>Function name</b>	spi_quad_disable
<b>Function prototype</b>	spi_quad_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable quad wire SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-881. Function spi\_quad\_write\_enable**

<b>Function name</b>	spi_quad_write_enable
<b>Function prototype</b>	void spi_quad_write_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable quad wire SPI write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-882. Function spi\_quad\_read\_enable**

<b>Function name</b>	spi_quad_read_enable
<b>Function prototype</b>	void spi_quad_read_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable quad wire SPI read
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 quad wire read */
```

```
spi_quad_read_enable(SPI0);
```

## spi\_i2s\_flag\_get

The description of spi\_i2s\_flag\_get is shown as below:

**Table 3-883. Function spi\_i2s\_flag\_get**

<b>Function name</b>	spi_i2s_flag_get
<b>Function prototype</b>	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);
<b>Function descriptions</b>	get SPI and I2S flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>flag</b>	SPI/I2S flag status
<i>SPI_FLAG_TBE</i>	transmit buffer empty flag
<i>SPI_FLAG_RBNE</i>	receive buffer not empty flag
<i>SPI_FLAG_TRANS</i>	transmit on-going flag
<i>SPI_I2S_INT_FLAG_RXORERR</i>	receive overrun error flag
<i>SPI_FLAG_CONFERR</i>	mode config error flag
<i>SPI_FLAG_CRCERR</i>	CRC error flag

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

Example:

```
/* get SPI0 transmit buffer empty flag status */
while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

## spi\_i2s\_interrupt\_enable

The description of spi\_i2s\_interrupt\_enable is shown as below:

**Table 3-884. Function spi\_i2s\_interrupt\_enable**

<b>Function name</b>	spi_i2s_interrupt_enable
<b>Function prototype</b>	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	enable SPI and I2S interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>interrupt</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-885. Function spi\_i2s\_interrupt\_disable**

<b>Function name</b>	spi_i2s_interrupt_disable
<b>Function prototype</b>	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	disable SPI and I2S interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>interrupt</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-886. Function spi\_i2s\_interrupt\_flag\_get**

<b>Function name</b>	spi_i2s_interrupt_flag_get
<b>Function prototype</b>	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	get SPI and I2S interrupt status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1



Input parameter{in}	
<b>interrupt</b>	SPI/I2S interrupt flag status
<i>SPI_I2S_INT_FLAG_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_FLAG_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_FLAG_RXORERR</i>	overrun interrupt
<i>SPI_INT_FLAG_CONFERR</i>	config error interrupt
<i>SPI_INT_FLAG_CRCERR</i>	CRC error interrupt
<i>I2S_INT_FLAG_TXURERR</i>	underrun error interrupt
<i>SPI_I2S_INT_FLAG_FORMATERR</i>	format error interrupt
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```

/* get SPI0 transmit buffer empty interrupt status */
if(RESET != spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE)){
    while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
    spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
}

```

## 3.26. 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 [3.26.1](#), the SQPI firmware functions are introduced in chapter [3.26.2](#).

### 3.26.1. Descriptions of Peripheral registers

SQPI registers are listed in the table shown as below:

**Table 3-887. SQPI Registers**

Registers	Descriptions
SQPI_INIT	SQPI initial register

Registers	Descriptions
SQPI_RCMD	SQPI read command register
SQPI_WCMD	SQPI write command register
SQPI_IDL	SQPI ID low register
SQPI_IDH	SQPI ID high register

### 3.26.2. Descriptions of Peripheral functions

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

**Table 3-888. 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 3-889. 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 3-890. Function sqpi\_deinit**

Function name	sqpi_deinit
Function prototype	void sqpi_deinit(void);
Function descriptions	reset SQPI peripheral

<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset SQPI */
```

```
sqpi_deinit();
```

### sqpi\_struct\_para\_init

The description of sqpi\_struct\_para\_init is shown as below:

**Table 3-891. Function sqpi\_struct\_para\_init**

<b>Function name</b>	sqpi_struct_para_init
<b>Function prototype</b>	void sqpi_struct_para_init(sqpi_parameter_struct* sqpi_struct);
<b>Function descriptions</b>	initialize the parameters of SQPI struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
*sqpi_struct	a sqpi_parameter_struct address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-892. Function sqpi\_init**

<b>Function name</b>	sqpi_init
<b>Function prototype</b>	void sqpi_init(sqpi_parameter_struct* sqpi_struct);
<b>Function descriptions</b>	initialize SQPI peripheral parameter

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sqi_struct</b>	SPI parameter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-889. sqpi_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-893. Function sqpi\_read\_id\_command**

<b>Function name</b>	sqpi_read_id_command
<b>Function prototype</b>	void sqpi_read_id_command (void);
<b>Function descriptions</b>	send SQPI read ID command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* send SQPI read ID command */

sqpi_read_id_command ();

```

## sqpi\_special\_command

The description of sqpi\_special\_command is shown as below:

**Table 3-894. Function sqpi\_special\_command**

<b>Function name</b>	sqpi_special_command
<b>Function prototype</b>	void sqpi_special_command(void);
<b>Function descriptions</b>	send SQPI special command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* send SQPI special command */
sqpi_special_command ();
```

## sqpi\_read\_command\_config

The description of sqpi\_read\_command\_config is shown as below:

**Table 3-895. Function sqpi\_read\_command\_config**

<b>Function name</b>	sqpi_read_command_config
<b>Function prototype</b>	void sqpi_read_command_config(uint32_t rmode, uint32_t rwaitcycle, uint32_t rcmd);
<b>Function descriptions</b>	configure SQPI read command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rmode</b>	SQPI read command mode
QSPI_MODE_SSQ	SSQ mode
QSPI_MODE_SSS	SSS mode
QSPI_MODE_SQQ	SQQ mode
QSPI_MODE_QQQ	QQQ mode
QSPI_MODE_SSD	SSD mode
QSPI_MODE_SDD	SDD mode
<b>Input parameter{in}</b>	
<b>rwaitcycle</b>	SQPI read wait cycle (0x00 – 0x1F)
<b>Input parameter{in}</b>	
<b>rcmd</b>	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 3-896. 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
QSPI_MODE_SSQ	SSQ mode
QSPI_MODE_SSS	SSS mode
QSPI_MODE_SQQ	SQQ mode
QSPI_MODE_QQQ	QQQ mode
QSPI_MODE_SSD	SSD mode
QSPI_MODE_SDD	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 3-897. Function sqpi\_low\_id\_receive**

<b>Function name</b>	sqpi_low_id_receive
<b>Function prototype</b>	uint32_t sqpi_low_id_receive(void);
<b>Function descriptions</b>	get low ID value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-898. Function sqpi\_low\_id\_receive**

<b>Function name</b>	sqpi_high_id_receive
<b>Function prototype</b>	uint32_t sqpi_high_id_receive(void);
<b>Function descriptions</b>	get high ID value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	32-bit ID value (0-0xFFFF)

Example:

```
/* get SQPI high ID value */
uint32_t val;
val = sqpi_high_id_receive ();
```

## 3.27. SYSCFG

The SYSCFG registers are listed in chapter [3.27.1](#), the SYSCFG firmware functions are introduced in chapter [3.27.2](#).

### 3.27.1. Descriptions of Peripheral registers

SYSCFG registers are listed in the table shown as below:

**Table 3-899. SYSCFG registers**

Registers	Descriptions
SYSCFG_CFG0	SYSCFG configuration register 0
SYSCFG_EXTISS0	EXTI sources selection register 0
SYSCFG_EXTISS1	EXTI sources selection register 1
SYSCFG_EXTISS2	EXTI sources selection register 2
SYSCFG_EXTISS3	EXTI sources selection register 3
SYSCFG_CPSCCTL	I/O compensation control register
SYSCFG_SECCFG	SYSCFG secure configuration register
SYSCFG_FPUINTMSK	FPU interrupt mask register
SYSCFG_CNSLOCK	SYSCFG CPU non-secure lock register
SYSCFG_CSLOCK	SYSCFG CPU secure lock register
SYSCFG_CFG1	SYSCFG configuration register 1
SYSCFG_SCS	SYSCFG SRAM1 control and status register
SYSCFG_SKEY	SYSCFG SRAM1 key register
SYSCFG_SWP0	SYSCFG SRAM1 write protection register 0
SYSCFG_SWP1	SYSCFG SRAM1 write protection register 1
SYSCFG_GSSACMD	SYSCFG GSSA command register
SYSCFG_TIMERCFG	TIMERx configuration register

### 3.27.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-900. SYSCFG firmware function**

Function name	Function description
syscfg_deinit	reset the SYSCFG registers
syscfg_exti_line_config	configure the GPIO pin as EXTI Line
syscfg_compensation_pwdn_mode_enable	enable the compensation cell



Function name	Function description
syscfg_compensation_pwdn_mode_disable	disable the compensation cell
syscfg_clock_access_security_config	configure SYSCFG clock control security
syscfg_classb_access_security_config	configure ClassB security
syscfg_sram1_access_security_config	configure SRAM1 security
syscfg_fpu_access_security_config	configure FPU security
syscfg_vtor_ns_write_disable	disable VTOR_NS register write
syscfg_mpu_ns_write_disable	disable Non-secure MPU registers write
syscfg_vtors_aircr_write_disable	disable VTOR_S register PRIS and BFHFNMINS bits in the AIRCR register write
syscfg_mpu_s_write_disable	disable secure MPU registers writes
syscfg_sau_write_disable	disable SAU registers write
syscfg_lock_config	connect TIMER0/15/16 break input to the selected parameter
syscfg_gssacmd_write_data	write data to GSSA
syscfg_sram1_erase	enable sram1 erase
syscfg_sram1_unlock	unlock the write protection of the SRAM1ERS bit in the SYSCFG_SCS register
syscfg_sram1_lock	lock the write protection of the SRAM1ERS bit in the SYSCFG_SCS register
syscfg_sram1_write_protect_0_31	SRAM1 write protect range from page0 to page31
syscfg_sram1_write_protect_32_63	SRAM1 write protect range from page32 to page63
syscfg_compensation_ready_flag_get	get the compensation ready flag
syscfg_sram1_bsy_flag_get	get SRAM1 erase busy flag
syscfg_fpu_interrupt_enable	enable floating point unit interrupt
syscfg_fpu_interrupt_disable	disable floating point unit interrupt

## syscfg\_deinit

The description of syscfg\_deinit is shown as below:

**Table 3-901. Function syscfg\_deinit**

Function name	syscfg_deinit
Function prototype	void syscfg_deinit(void);
Function descriptions	reset the SYSCFG registers
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* reset SYSCFG */
```

```
syscfg_deinit();
```

### syscfg\_exti\_line\_config

The description of syscfg\_exti\_line\_config is shown as below:

**Table 3-902. Function syscfg\_exti\_line\_config**

<b>Function name</b>	syscfg_exti_line_config
<b>Function prototype</b>	void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);
<b>Function descriptions</b>	configure the GPIO pin as EXTI Line
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exti_port</b>	specify the GPIO port used in EXTI
EXTI_SOURCE_GPIOx	EXTI GPIO port (x = A, B, C)
<b>Input parameter{in}</b>	
<b>exti_pin</b>	specify the EXTI line
EXTI_SOURCE_PINx	EXTI GPIO pin (GPIOA x = 0..15, GPIOB x = 0..15, GPIOC x = 0..8)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the GPIO pin as EXTI Line */
```

```
syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN1);
```

### syscfg\_compensation\_pwdn\_mode\_enable

The description of syscfg\_compensation\_pwdn\_mode\_enable is shown as below:

**Table 3-903. Function syscfg\_compensation\_pwdn\_mode\_enable**

<b>Function name</b>	syscfg_compensation_pwdn_mode_enable
<b>Function prototype</b>	void syscfg_compensation_pwdn_mode_enable(void);
<b>Function descriptions</b>	enable the compensation cell
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* enable the compensation cell */
```

```
syscfg_compensation_pwdn_mode_enable();
```

### syscfg\_compensation\_pwdn\_mode\_disable

The description of syscfg\_compensation\_pwdn\_mode\_disable is shown as below:

**Table 3-904. Function syscfg\_compensation\_pwdn\_mode\_disable**

<b>Function name</b>	syscfg_compensation_pwdn_mode_disable
<b>Function prototype</b>	void syscfg_compensation_pwdn_mode_disable (void)
<b>Function descriptions</b>	disable the compensation cell
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the compensation cell */
```

```
syscfg_compensation_pwdn_mode_disable();
```

### syscfg\_clock\_access\_security\_config

The description of syscfg\_clock\_access\_security\_config is shown as below:

**Table 3-905. Function syscfg\_clock\_access\_security\_config**

<b>Function name</b>	syscfg_clock_access_security_config
<b>Function prototype</b>	void syscfg_clock_access_security_config(uint32_t access_mode);
<b>Function descriptions</b>	configure SYSCFG clock control security
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>access_mode</b>	secure access or secure and non-secure access
SYSCFG_SECURE_ACCESS	SYSCFG configuration clock in RCU registers can be written by secure access only
SYSCFG_SECURE_N	SYSCFG configuration clock in RCU registers can be written by secure and

<code>ONSECURE_ACCESS</code>	nonsecure access
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SYSCFG clock control security */
```

```
syscfg_clock_access_security_config(SYSCFG_SECURE_ACCESS);
```

## **syscfg\_classb\_access\_security\_config**

The description of `syscfg_classb_access_security_config` is shown as below:

**Table 3-906. Function `syscfg_classb_access_security_config`**

<b>Function name</b>	<code>syscfg_classb_access_security_config</code>
<b>Function prototype</b>	<code>void syscfg_classb_access_security_config(uint32_t access_mode);</code>
<b>Function descriptions</b>	configure ClassB security
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>access_mode</b>	secure access or secure and non-secure access
<code>CLASSB_SECURE_ACCESS</code>	SYSCFG_CFG1 register can be written by secure access only
<code>CLASSB_SECURE_NONSECURE_ACCESS</code>	SYSCFG_CFG1 register can be written by secure and non-secure access
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ClassB security */
```

```
syscfg_classb_access_security_config(CLASSB_SECURE_ACCESS);
```

## **syscfg\_sram1\_access\_security\_config**

The description of `syscfg_sram1_access_security_config` is shown as below:

**Table 3-907. Function `syscfg_sram1_access_security_config`**

<b>Function name</b>	<code>syscfg_sram1_access_security_config</code>
<b>Function prototype</b>	<code>void syscfg_sram1_access_security_config(uint32_t access_mode)</code>
<b>Function descriptions</b>	configure SRAM1 security
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>access_mode</b>	secure access or secure and non-secure access
<i>SRAM1_SECURE_ACCESS</i>	SYSCFG_SKEY, SYSCFG_SCS and SYSCFG_SWPx registers can be written by secure only
<i>SRAM1_SECURE_ACCESS</i>	SYSCFG_SKEY, SYSCFG_SCS and SYSCFG_SWPx registers can be written by secure and non-secure access
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SRAM1 security */
```

```
syscfg_sram1_access_security_config(SRAM1_SECURE_ACCESS);
```

### syscfg\_fpu\_access\_security\_config

The description of syscfg\_fpu\_access\_security\_config is shown as below:

**Table 3-908. Function syscfg\_fpu\_access\_security\_config**

<b>Function name</b>	syscfg_fpu_access_security_config
<b>Function prototype</b>	void syscfg_fpu_access_security_config(uint32_t access_mode);
<b>Function descriptions</b>	configure FPU security
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>access_mode</b>	secure access or secure and non-secure access
<i>FPU_SECURE_ACCESS</i>	SYSCFG_FPUINTMSK register can be written by secure access only
<i>FPU_SECURE_NONSECURE_ACCESS</i>	SYSCFG_FPUINTMSK register can be written by secure and non-secure access
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure FPU security */
```

```
syscfg_fpu_access_security_config(FPU_SECURE_ACCESS);
```

## syscfg\_vtor\_ns\_write\_disable

The description of syscfg\_vtor\_ns\_write\_disable is shown as below:

**Table 3-909. Function syscfg\_vtor\_ns\_write\_disable**

<b>Function name</b>	syscfg_vtor_ns_write_disable
<b>Function prototype</b>	void syscfg_vtor_ns_write_disable(void);
<b>Function descriptions</b>	disable VTOR_NS register write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable VTOR_NS register write */
syscfg_vtor_ns_write_disable();
```

## syscfg\_mpu\_ns\_write\_disable

The description of syscfg\_mpu\_ns\_write\_disable is shown as below:

**Table 3-910. Function syscfg\_mpu\_ns\_write\_disable**

<b>Function name</b>	syscfg_mpu_ns_write_disable
<b>Function prototype</b>	void syscfg_mpu_ns_write_disable(void)
<b>Function descriptions</b>	disable Non-secure MPU registers write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable Non-secure MPU registers write */
syscfg_mpu_ns_write_disable();
```

## syscfg\_vtors\_aircr\_write\_disable

The description of syscfg\_vtors\_aircr\_write\_disable is shown as below:

**Table 3-911. Function syscfg\_vtors\_aircr\_write\_disable**

<b>Function name</b>	syscfg_vtors_aircr_write_disable
<b>Function prototype</b>	void syscfg_vtors_aircr_write_disable(void)
<b>Function descriptions</b>	disable VTOR_S register PRIS and BFHFNMINS bits in the AIRCR register write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable VTOR_S register PRIS and BFHFNMINS bits in the AIRCR register write */
syscfg_vtors_aircr_write_disable();
```

## syscfg\_mpu\_s\_write\_disable

The description of syscfg\_mpu\_s\_write\_disable is shown as below:

**Table 3-912. Function syscfg\_mpu\_s\_write\_disable**

<b>Function name</b>	syscfg_mpu_s_write_disable
<b>Function prototype</b>	void syscfg_mpu_s_write_disable(void);
<b>Function descriptions</b>	disable secure MPU registers writes
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable secure MPU registers writes */
syscfg_mpu_s_write_disable();
```

## syscfg\_sau\_write\_disable

The description of syscfg\_sau\_write\_disable is shown as below:

**Table 3-913. Function sau\_write\_disable**

<b>Function name</b>	syscfg_sau_write_disable
<b>Function prototype</b>	void syscfg_sau_write_disable(void);
<b>Function descriptions</b>	disable SAU registers write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SAU registers write */
syscfg_sau_write_disable();
```

## syscfg\_lock\_config

The description of syscfg\_lock\_config is shown as below:

**Table 3-914. Function syscfg\_lock\_config**

<b>Function name</b>	syscfg_lock_config
<b>Function prototype</b>	void syscfg_lock_config(uint32_t syscfg_lock);
<b>Function descriptions</b>	connect TIMER0/15/16 break input to the selected parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_lock</b>	specify the parameter to be connected
<b>SYSCFG_LOCK_LOCKUP</b>	Cortex-M3 lockup output connected to the break input
<b>SYSCFG_LOCK_LVD</b>	LVD interrupt connected to the break input
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* connect TIMER0/15/16 break input to the selected parameter */
syscfg_lock_config(SYSCFG_LOCK_LOCKUP);
```



## syscfg\_gssacmd\_write\_data

The description of syscfg\_gssacmd\_write\_data is shown as below:

**Table 3-915. Function syscfg\_gssacmd\_write\_data**

<b>Function name</b>	syscfg_gssacmd_write_data
<b>Function prototype</b>	void syscfg_gssacmd_write_data(uint32_t data);
<b>Function descriptions</b>	write data to GSSA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data</b>	the data to be written in to GSSA(0x00-0xFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* write data to GSSA */
syscfg_gssacmd_write_data(0x0C);
```

## syscfg\_sram1\_erase

The description of syscfg\_sram1\_erase is shown as below:

**Table 3-916. Function syscfg\_sram1\_erase**

<b>Function name</b>	syscfg_sram1_erase
<b>Function prototype</b>	ErrStatus syscfg_sram1_erase(void);
<b>Function descriptions</b>	enable sram1 erase
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* enable sram1 erase */
ErrStatus state;
state = syscfg_sram1_erase();
```

## syscfg\_sram1\_unlock

The description of syscfg\_sram1\_unlock is shown as below:

**Table 3-917. Function syscfg\_sram1\_unlock**

<b>Function name</b>	syscfg_sram1_unlock
<b>Function prototype</b>	void syscfg_sram1_unlock(void);
<b>Function descriptions</b>	unlock the write protection of the SRAM1ERS bit in the SYSCFG_SCS register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* unlock the write protection of the SRAM1ERS bit in the SYSCFG_SCS register */
syscfg_sram1_unlock();
```

## syscfg\_sram1\_lock

The description of syscfg\_sram1\_lock is shown as below:

**Table 3-918. Function syscfg\_sram1\_lock**

<b>Function name</b>	syscfg_sram1_lock
<b>Function prototype</b>	void syscfg_sram1_lock(void);
<b>Function descriptions</b>	lock the write protection of the SRAM1ERS bit in the SYSCFG_SCS register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock the write protection of the SRAM1ERS bit in the SYSCFG_SCS register */
syscfg_sram1_lock();
```

## syscfg\_sram1\_write\_protect\_0\_31

The description of syscfg\_sram1\_write\_protect\_0\_31 is shown as below:

**Table 3-919. Function syscfg\_sram1\_write\_protect\_0\_31**

<b>Function name</b>	syscfg_sram1_write_protect_0_31
<b>Function prototype</b>	void syscfg_sram1_write_protect_0_31(uint32_t page);
<b>Function descriptions</b>	SRAM1 write protect range from page0 to page31
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>page</b>	specify the page of SRAM1 to protect
<i>SRAM1_PAGEx_WRITE_PROTECT</i>	enable write protection of SRAM1 page x (x=0,1,...31)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SRAM1 write protect page1 */
```

```
syscfg_sram1_write_protect_0_31(SRAM1_PAGE1_WRITE_PROTECT);
```

## syscfg\_sram1\_write\_protect\_32\_63

The description of syscfg\_sram1\_write\_protect\_32\_63 is shown as below:

**Table 3-920. Function syscfg\_sram1\_write\_protect\_32\_63**

<b>Function name</b>	syscfg_sram1_write_protect_32_63
<b>Function prototype</b>	void syscfg_sram1_write_protect_32_63(uint32_t page);
<b>Function descriptions</b>	SRAM1 write protect range from page32 to page63
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>page</b>	specify the page of SRAM1 to protect
<i>SRAM1_PAGEx_WRITE_PROTECT</i>	enable write protection of SRAM1 page x (x=32,33,...63)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SRAM1 write protect page32 */
```

syscfg\_sram1\_write\_protect\_32\_63(SRAM1\_PAGE32\_WRITE\_PROTECT);

## syscfg\_compensation\_ready\_flag\_get

The description of syscfg\_compensation\_ready\_flag\_get is shown as below:

**Table 3-921. Function syscfg\_compensation\_ready\_flag\_get**

<b>Function name</b>	syscfg_compensation_ready_flag_get
<b>Function prototype</b>	FlagStatus syscfg_compensation_ready_flag_get(void);
<b>Function descriptions</b>	get the compensation ready flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	RESET or SET

Example:

```
/* get the compensation ready flag */
syscfg_compensation_ready_flag_get();
```

## syscfg\_sram1\_bsy\_flag\_get

The description of syscfg\_sram1\_bsy\_flag\_get is shown as below:

**Table 3-922. Function syscfg\_sram1\_bsy\_flag\_get**

<b>Function name</b>	syscfg_sram1_bsy_flag_get
<b>Function prototype</b>	FlagStatus syscfg_sram1_bsy_flag_get(void);
<b>Function descriptions</b>	get SRAM1 erase busy flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET of RESET

Example:

```
/* get SRAM1 erase busy flag */
FlagStatus state;
```

```
State = syscfg_sram1_bsy_flag_get();
```

## syscfg\_fpu\_interrupt\_enable

The description of syscfg\_fpu\_interrupt\_enable is shown as below:

**Table 3-923. Function syscfg\_fpu\_interrupt\_enable**

<b>Function name</b>	syscfg_fpu_interrupt_enable
<b>Function prototype</b>	void syscfg_fpu_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable floating point unit interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type
INVALID_OPERATION_INT	invalid operation Interrupt
DEVIDE_BY_ZERO_INT	divide-by-zero interrupt
UNDERFLOW_INT	underflow interrupt
OVERFLOW_INT	overflow interrupt
INPUT_ABNORMAL_INTERRUPT	input abnormal interrupt
INEXACT_INT	inexact interrupt (interrupt disable at reset)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable floating point unit interrupt */
syscfg_fpu_interrupt_enable(INVALID_OPERATION_INT);
```

## syscfg\_fpu\_interrupt\_disable

The description of syscfg\_fpu\_interrupt\_disable is shown as below:

**Table 3-924. Function syscfg\_fpu\_interrupt\_disable**

<b>Function name</b>	syscfg_fpu_interrupt_disable
<b>Function prototype</b>	void syscfg_fpu_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable floating point unit interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type

<i>INVALID_OPERATION_INT</i>	invalid operation Interrupt
<i>DEVIDE_BY_ZERO_INT</i>	divide-by-zero interrupt
<i>UNDERFLOW_INT</i>	underflow interrupt
<i>OVERFLOW_INT</i>	overflow interrupt
<i>INPUT_ABNORMAL_INT</i>	input abnormal interrupt
<i>INEXACT_INT</i>	inexact interrupt (interrupt disable at reset)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable floating point unit interrupt */
```

```
syscfg_fpu_interrupt_disable(INVALID_OPERATION_INT);
```

## 3.28. 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), general level0 timer (TIMERx, x=1, 2, 3, 4), general level4 timer (TIMERx, x=15, 16), basic timer (TIMERx, x=5). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.28.1](#), the TIMER firmware functions are introduced in chapter [3.28.2](#).

### 3.28.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

**Table 3-925. TIMERx Registers**

Registers	Descriptions
TIMER_CTL0	Control register 0
TIMER_CTL1	Control register 1
TIMER_SMCFG	Slave mode configuration register
TIMER_DMAINTEN	DMA and interrupt enable register
TIMER_INTF	Interrupt flag register
TIMER_SWEVG	Software event generation register
TIMER_CHCTL0	Channel control register 0
TIMER_CHCTL1	Channel control register 1
TIMER_CHCTL2	Channel control register 2

Registers	Descriptions
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

## 3.28.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-926. TIMEx firmware function**

Function name	Function description
timer_deinit	deinit a timer
timer_struct_para_init	initialize the parameters of TIMER init parameter struct with the default values
timer_init	initialize TIMER counter
timer_enable	enable a timer
timer_disable	disable a timer
timer_auto_reload_shadow_enable	enable the auto reload shadow function
timer_auto_reload_shadow_disable	disable the auto reload shadow function
timer_update_event_enable	enable the update event
timer_update_event_disable	disable the update event
timer_counter_alignment	set TIMER counter alignment mode
timer_counter_up_direction	set TIMER counter up direction
timer_counter_down_direction	set TIMER counter down direction
timer_prescaler_config	configure TIMER prescaler
timer_repetition_value_config	configure TIMER repetition register value
timer_autoreload_value_config	configure TIMER autoreload register value
timer_counter_value_config	configure TIMER counter register value
timer_counter_read	read TIMER counter value
timer_prescaler_read	read TIMER prescaler value
timer_single_pulse_mode_config	configure TIMER single pulse mode
timer_update_source_config	configure TIMER update source
timer_dma_enable	enable the TIMER DMA
timer_dma_disable	disable the TIMER DMA

Function name	Function description
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
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



Function name	Function description
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 3-927. 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 3-928. 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 3-929. Structure timer\_oc\_parameter\_struct**

Member name	Function description
outputstate	channel output state (TIMER_CCX_ENABLE, TIMER_CCX_DISABLE)
outputnstate	channel complementary output state (TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE)
ocpolarity	channel output polarity (TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW)
ocnpolarity	channel complementary output polarity (TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW)
ocidlestate	idle state of channel output (TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH)
ocnidlestate	idle state of channel complementary output (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH)

## Structure timer\_ic\_parameter\_struct

**Table 3-930. 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 3-931. Function timer\_deinit**

<b>Function name</b>	timer_deinit
<b>Function prototype</b>	void timer_deinit(uint32_t timer_periph);
<b>Function descriptions</b>	deinit a TIMER
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..5, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset TIMER0 */
timer_deinit(TIMER0);
```

## timer\_struct\_para\_init

The description of timer\_struct\_para\_init is shown as below:

**Table 3-932. Function timer\_struct\_para\_init**

<b>Function name</b>	timer_struct_para_init
<b>Function prototype</b>	void timer_struct_para_init(timer_parameter_struct* initpara);
<b>Function descriptions</b>	initialize the parameters of TIMER init parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>initpara</b>	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-927. Structure timer parameter struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-933. Function timer\_init**

<b>Function name</b>	timer_init
<b>Function prototype</b>	void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);
<b>Function descriptions</b>	initialize TIMER counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..5, 15, 16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>initpara</b>	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-927. Structure timer_parameter_struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-934. Function timer\_enable**

<b>Function name</b>	timer_enable
<b>Function prototype</b>	void timer_enable(uint32_t timer_periph);

<b>Function descriptions</b>	enable a timer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..5, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 */
timer_enable(TIMER0);
```

## timer\_disable

The description of timer\_disable is shown as below:

**Table 3-935. Function timer\_disable**

<b>Function name</b>	timer_disable
<b>Function prototype</b>	void timer_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable a timer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..5, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 */
timer_disable(TIMER0);
```

## timer\_auto\_reload\_shadow\_enable

The description of timer\_auto\_reload\_shadow\_enable is shown as below:

**Table 3-936. Function timer\_auto\_reload\_shadow\_enable**

<b>Function name</b>	timer_auto_reload_shadow_enable
<b>Function prototype</b>	void timer_auto_reload_shadow_enable(uint32_t timer_periph);

<b>Function descriptions</b>	enable the auto reload shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..5, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-937. Function timer\_auto\_reload\_shadow\_disable**

<b>Function name</b>	timer_auto_reload_shadow_disable
<b>Function prototype</b>	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable the auto reload shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..5, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-938. Function timer\_update\_event\_enable**

<b>Function name</b>	timer_update_event_enable
<b>Function prototype</b>	void timer_update_event_enable(uint32_t timer_periph);

<b>Function descriptions</b>	enable the update event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..5, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-939. Function timer\_update\_event\_disable**

<b>Function name</b>	timer_update_event_disable
<b>Function prototype</b>	void timer_update_event_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable the update event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..5, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-940. Function timer\_counter\_alignment**

<b>Function name</b>	timer_counter_alignment
<b>Function prototype</b>	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);

<b>Function descriptions</b>	set TIMER counter alignment mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>aligned</b>	alignment mode
<i>TIMER_COUNTER_EDGE</i>	No center-aligned mode (edge-aligned mode). The direction of the counter is specified by the DIR bit.
<i>TIMER_COUNTER_COUNTER_DOWN</i>	Center-aligned and counting down assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in <i>TIMERx_CHCTL0</i> register). Only when the counter is counting down, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_COUNTER_UP</i>	Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in <i>TIMERx_CHCTL0</i> register). Only when the counter is counting up, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_COUNTER_BOTH</i>	Center-aligned and counting up/down assert mode. The counter counts under center-aligned and channel is configured in output mode (CHxMS=00 in <i>TIMERx_CHCTL0</i> register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set TIMER0 counter center-aligned and counting up assert mode */
timer_counter_alignment(TIMER0, TIMER_COUNTER_COUNTER_UP);
```

## timer\_counter\_up\_direction

The description of timer\_counter\_up\_direction is shown as below:

**Table 3-941. Function timer\_counter\_up\_direction**

<b>Function name</b>	timer_counter_up_direction
<b>Function prototype</b>	void timer_counter_up_direction(uint32_t timer_periph);
<b>Function descriptions</b>	set TIMER counter up direction
<b>Precondition</b>	set TIMER counter no center-aligned mode (edge-aligned mode)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral



<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-942. timer\_counter\_down\_direction**

<b>Function name</b>	timer_counter_down_direction
<b>Function prototype</b>	void timer_counter_down_direction(uint32_t timer_periph);
<b>Function descriptions</b>	set TIMER counter down direction
<b>Precondition</b>	set TIMER counter no center-aligned mode (edge-aligned mode)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-943. Function timer\_prescaler\_config**

<b>Function name</b>	timer_prescaler_config
<b>Function prototype</b>	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint32_t pscreload);
<b>Function descriptions</b>	configure TIMER prescaler
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..5,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>prescaler</b>	prescaler value (0~65535)
<b>Input parameter{in}</b>	
<b>pscreload</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-944. Function timer\_repetition\_value\_config**

<b>Function name</b>	timer_repetition_value_config
<b>Function prototype</b>	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
<b>Function descriptions</b>	configure TIMER repetition register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>repetition</b>	the counter repetition value (0~255)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-945. Function timer\_autoreload\_value\_config**

<b>Function name</b>	timer_autoreload_value_config
<b>Function prototype</b>	void timer_autoreload_value_config(uint32_t timer_periph, uint32_t autoreload);
<b>Function descriptions</b>	configure TIMER autoreload register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..5, 15, 16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>autoreload</b>	the counter auto-reload value (0-0xFFFFFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER autoreload register value */
timer_autoreload_value_config(TIMERO0, 3000);
```

## timer\_counter\_value\_config

The description of timer\_counter\_value\_config is shown as below:

**Table 3-946. Function timer\_counter\_value\_config**

<b>Function name</b>	timer_counter_value_config
<b>Function prototype</b>	void timer_counter_value_config(uint32_t timer_periph, uint32_t counter);
<b>Function descriptions</b>	configure TIMER counter register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..5, 15, 16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>counter</b>	the counter value (0-0xFFFFFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-947. Function timer\_counter\_read**

<b>Function name</b>	timer_counter_read
<b>Function prototype</b>	uint32_t timer_counter_read(uint32_t timer_periph);
<b>Function descriptions</b>	read TIMER counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..5, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	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 3-948. Function timer\_prescaler\_read**

<b>Function name</b>	timer_prescaler_read
<b>Function prototype</b>	uint16_t timer_prescaler_read(uint32_t timer_periph);
<b>Function descriptions</b>	read TIMER prescaler value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..5, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

<b>uint16_t</b>	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 3-949. Function timer\_single\_pulse\_mode\_config**

<b>Function name</b>	timer_single_pulse_mode_config
<b>Function prototype</b>	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
<b>Function descriptions</b>	configure TIMER single pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..5, 15, 16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>spmode</b>	pulse mode
<i>TIMER_SP_MODE_SINGLE</i>	single pulse mode
<i>TIMER_SP_MODE_REPETITIVE</i>	repetitive pulse mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-950. Function timer\_update\_source\_config**

<b>Function name</b>	timer_update_source_config
<b>Function prototype</b>	void timer_update_source_config(uint32_t timer_periph, uint32_t update);

<b>Function descriptions</b>	configure TIMER update source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> ( <i>x</i> =0..5, 15, 16)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>update</b>	update source
<i>TIMER_UPDATE_SRC_GLOBAL</i>	Any of the following events generate an update interrupt or DMA request:  – The UPG bit is set  – The counter generates an overflow or underflow event  – The slave mode controller generates an update event
<i>TIMER_UPDATE_SRC_REGULAR</i>	Only counter overflow/underflow generates an update interrupt or DMA request.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-951. Function timer\_dma\_enable**

<b>Function name</b>	timer_dma_enable
<b>Function prototype</b>	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
<b>Function descriptions</b>	enable the TIMER DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma</b>	timer DMA source enable
<i>TIMER_DMA_UPD</i>	update DMA enable, <i>TIMERx</i> ( <i>x</i> =0..5,15,16)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable, <i>TIMERx</i> ( <i>x</i> =0..4,,15,16)

<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable, <i>TIMERx</i> ( <i>x</i> =0,15,16)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable, <i>TIMERx</i> ( <i>x</i> =0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-952. Function timer\_dma\_disable**

<b>Function name</b>	timer_dma_disable
<b>Function prototype</b>	void timer_dma_disable (uint32_t timer_periph, uint16_t dma);
<b>Function descriptions</b>	disable the TIMER DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma</b>	timer DMA source disable
<i>TIMER_DMA_UPD</i>	update DMA disable, <i>TIMERx</i> ( <i>x</i> =0..5,15,16)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA disable, <i>TIMERx</i> ( <i>x</i> =0..4,15,16)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA disable, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA disable, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA disable, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMA_CMTD</i>	commutation DMA request disable, <i>TIMERx</i> ( <i>x</i> =0,15,16)
<i>TIMER_DMA_TRGD</i>	trigger DMA disable, <i>TIMERx</i> ( <i>x</i> =0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-953. Function timer\_channel\_dma\_request\_source\_select**

<b>Function name</b>	timer_channel_dma_request_source_select
<b>Function prototype</b>	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
<b>Function descriptions</b>	channel DMA request source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>dma_request</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-954. Function timer\_dma\_transfer\_config**

<b>Function name</b>	timer_dma_transfer_config
<b>Function prototype</b>	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
<b>Function descriptions</b>	configure the TIMER DMA transfer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	



<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma_baseaddr</b>	DMA transfer access start address
<i>TIMER_DMACFG_DMA TA_CTL0</i>	DMA transfer address is TIMER_CTL0, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA TA_CTL1</i>	DMA transfer address is TIMER_CTL1, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA TA_SMCFG</i>	DMA transfer address is TIMER_SMCFG, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA TA_DMAINTEN</i>	DMA transfer address is TIMER_DMAINTEN, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA TA_INTF</i>	DMA transfer address is TIMER_INTF, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA TA_SWEVG</i>	DMA transfer address is TIMER_SWEVG, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA TA_CHCTL0</i>	DMA transfer address is TIMER_CHCTL0, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA TA_CHCTL1</i>	DMA transfer address is TIMER_CHCTL1, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA TA_CHCTL2</i>	DMA transfer address is TIMER_CHCTL2, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA TA_CNT</i>	DMA transfer address is TIMER_CNT, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA TA_PSC</i>	DMA transfer address is TIMER_PSC, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA TA_CAR</i>	MA transfer address is TIMER_CAR, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA TA_CREP</i>	DMA transfer address is TIMER_CREP, TIMERx(x=0,15,16)
<i>TIMER_DMACFG_DMA TA_CH0CV</i>	DMA transfer address is TIMER_CH0CV, TIMERx(x=0..4,15,16)
<i>TIMER_DMACFG_DMA TA_CH1CV</i>	DMA transfer address is TIMER_CH1CV, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA TA_CH2CV</i>	DMA transfer address is TIMER_CH2CV, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA TA_CH3CV</i>	DMA transfer address is TIMER_CH3CV, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA TA_CCHP</i>	DMA transfer address is TIMER_CCHP, TIMERx(x=0,15,16)
<i>TIMER_DMACFG_DMA TA_DMACFG</i>	DMA transfer address is TIMER_DMACFG, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMA</i>	DMA transfer address is TIMER_DMATB, TIMERx(x=0..4)

<i>TA_DMATB</i>	
<b>Input parameter{in}</b>	
<b>dma_lenth</b>	DMA transfer count
<i>TIMER_DMACFG_DMA TC_xTRANSFER</i>	x=1..18, DMA transfer x time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-955. Function timer\_event\_software\_generate**

<b>Function name</b>	timer_event_software_generate
<b>Function prototype</b>	void timer_event_software_generate(uint32_t timer_periph, uint16_t event);
<b>Function descriptions</b>	software generate events
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>event</b>	the timer software event generation sources
<i>TIMER_EVENT_SRC_UPG</i>	update event, TIMERx(x=0..5,15,16)
<i>TIMER_EVENT_SRC_C0G</i>	channel 0 capture or compare event generation, TIMERx(x=0..4,15,16)
<i>TIMER_EVENT_SRC_C1G</i>	channel 1 capture or compare event generation, TIMERx(x=0..4)
<i>TIMER_EVENT_SRC_C2G</i>	channel 2 capture or compare event generation, TIMERx(x=0..4)
<i>TIMER_EVENT_SRC_C3G</i>	channel 3 capture or compare event generation, TIMERx(x=0..4)
<i>TIMER_EVENT_SRC_CMTG</i>	channel commutation event generation, TIMERx(x=0,15,16)

<i>TIMER_EVENT_SRC_T RGG</i>	trigger event generation, TIMERx(x=0..4)
<i>TIMER_EVENT_SRC_B RKG</i>	break event generation, TIMERx(x=0,15,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-956. Function timer\_break\_struct\_para\_init**

<b>Function name</b>	timer_break_struct_para_init
<b>Function prototype</b>	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
<b>Function descriptions</b>	initialize the parameters of TIMER break parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>breakpara</b>	TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-928. Structure timer break parameter struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-957. Function timer\_break\_config**

<b>Function name</b>	timer_break_config
<b>Function prototype</b>	void timer_break_config(uint32_t timer_periph, timer_break_parameter_struct* breakpara);
<b>Function descriptions</b>	configure TIMER break function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>breakpara</b>	TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-928. Structure timer_break_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-958. Function timer\_break\_enable**

<b>Function name</b>	timer_break_enable
<b>Function prototype</b>	void timer_break_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable TIMER break function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in

	TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 break function*/
```

```
timer_break_enable (TIMER0);
```

## timer\_break\_disable

The description of timer\_break\_disable is shown as below:

**Table 3-959. Function timer\_break\_disable**

<b>Function name</b>	timer_break_disable
<b>Function prototype</b>	void timer_break_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER break function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-960. Function timer\_automatic\_output\_enable**

<b>Function name</b>	timer_automatic_output_enable
<b>Function prototype</b>	void timer_automatic_output_enable(uint32_t timer_periph);

<b>Function descriptions</b>	enable TIMER output automatic function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-961. Function timer\_automatic\_output\_disable**

<b>Function name</b>	timer_automatic_output_disable
<b>Function prototype</b>	void timer_automatic_output_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER output automatic function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-962. Function timer\_primary\_output\_config**

<b>Function name</b>	timer_primary_output_config
<b>Function prototype</b>	void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue);
<b>Function descriptions</b>	configure TIMER primary output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-963. Function timer\_channel\_control\_shadow\_config**

<b>Function name</b>	timer_channel_control_shadow_config
<b>Function prototype</b>	void timer_channel_control_shadow_config(uint32_t timer_periph, ControlStatus newvalue);
<b>Function descriptions</b>	channel commutation control shadow register enable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-

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 3-964. 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}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
Input parameter{in}	
<b>ccuctl</b>	channel control shadow register update control
<i>TIMER_UPDATECTL_CCU</i>	the shadow registers update by when CMTG bit is set
<i>TIMER_UPDATECTL_CUTRI</i>	the shadow registers update by when CMTG bit is set or an rising edge of TRGI occurs
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel control shadow register update when CMTG bit is set */
```

```
timer_channel_control_shadow_update_config(TIMER0, TIMER_UPDATECTL_CCU);
```

## timer\_channel\_output\_struct\_para\_init

The description of timer\_channel\_output\_struct\_para\_init is shown as below:

**Table 3-965. Function timer\_channel\_output\_struct\_para\_init**

Function name	timer_channel_output_struct_para_init
Function prototype	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpa);



<b>Function descriptions</b>	initialize the parameters of TIMER channel output parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ocpara</b>	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-929. Structure timer_oc_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-966. Function timer\_channel\_output\_config**

<b>Function name</b>	timer_channel_output_config
<b>Function prototype</b>	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);
<b>Function descriptions</b>	configure TIMER channel output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4))
<b>Input parameter{in}</b>	
<b>ocpara</b>	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-929. Structure timer_oc_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

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 3-967. Function timer\_channel\_output\_mode\_config**

Function name	timer_channel_output_mode_config
Function prototype	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);
Function descriptions	configure TIMER channel output compare mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMERx(x=0..4,15,16))
TIMER_CH_1	TIMER channel 1 (TIMERx(x=0..4))
TIMER_CH_2	TIMER channel 2 (TIMERx(x=0..4))
TIMER_CH_3	TIMER channel 3 (TIMERx(x=0..4))
Input parameter{in}	
ocmode	channel output compare mode
TIMER_OC_MODE_TIMING	timing mode
TIMER_OC_MODE_ACTIVE	set the channel output
TIMER_OC_MODE_INACTIVE	clear the channel output

<i>CTIVE</i>	
<i>TIMER_OC_MODE_TOGGLE</i>	toggle on match
<i>TIMER_OC_MODE_LOW</i>	force low mode
<i>TIMER_OC_MODE_HIGH</i>	force high mode
<i>TIMER_OC_MODE_PWM0</i>	PWM mode 0
<i>TIMER_OC_MODE_PWM1</i>	PWM mode 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-968. Function timer\_channel\_output\_pulse\_value\_config**

<b>Function name</b>	timer_channel_output_pulse_value_config
<b>Function prototype</b>	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);
<b>Function descriptions</b>	configure TIMER channel output pulse value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4))
<b>Input parameter{in}</b>	
<b>pulse</b>	channel output pulse value (0~0xFFFFFFFF)
<b>Output parameter{out}</b>	

-	-
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 3-969. Function timer\_channel\_output\_shadow\_config**

<b>Function name</b>	timer_channel_output_shadow_config
<b>Function prototype</b>	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
<b>Function descriptions</b>	configure TIMER channel output shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4))
<b>Input parameter{in}</b>	
<b>ocshadow</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-970. Function timer\_channel\_output\_fast\_config**

<b>Function name</b>	timer_channel_output_fast_config
<b>Function prototype</b>	void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
<b>Function descriptions</b>	configure TIMER channel output fast function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4))
<b>Input parameter{in}</b>	
<b>ocfast</b>	channel output fast function
<i>TIMER_OC_FAST_ENABLE</i>	channel output fast function enable
<i>TIMER_OC_FAST_DISABLE</i>	channel output fast function disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-971. Function timer\_channel\_output\_clear\_config**

<b>Function name</b>	timer_channel_output_clear_config
<b>Function prototype</b>	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
<b>Function descriptions</b>	configure TIMER channel output clear function

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER periph
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4))
<b>Input parameter{in}</b>	
<b>occlear</b>	channel output clear function
<i>TIMER_OC_CLEAR_ENABLE</i>	channel output clear function enable
<i>TIMER_OC_CLEAR_DISABLE</i>	channel output clear function disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output clear function */
```

```
timer_channel_output_clear_config (TIMER0, TIMER_CH_0,  
TIMER_OC_CLEAR_ENABLE);
```

## timer\_channel\_output\_polarity\_config

The description of timer\_channel\_output\_polarity\_config is shown as below:

**Table 3-972. Function timer\_channel\_output\_polarity\_config**

<b>Function name</b>	timer_channel_output_polarity_config
<b>Function prototype</b>	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
<b>Function descriptions</b>	configure TIMER channel output polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured

<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4))
<b>Input parameter{in}</b>	
<b>ocpolarity</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output polarity */
```

```
timer_channel_output_polarity_config(TIMER0, TIMER_CH_0,  
TIMER_OC_POLARITY_HIGH);
```

## timer\_channel\_complementary\_output\_polarity\_config

The description of timer\_channel\_complementary\_output\_polarity\_config is shown as below:

**Table 3-973. Function timer\_channel\_complementary\_output\_polarity\_config**

<b>Function name</b>	timer_channel_complementary_output_polarity_config
<b>Function prototype</b>	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnpolarity);
<b>Function descriptions</b>	configure TIMER channel complementary output polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0, 15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0))
<b>Input parameter{in}</b>	
<b>ocpolarity</b>	channel complementary output polarity
<i>TIMER_OCN_POLARITY_HIGH</i>	channel complementary output polarity is high

<i>TIMER_OCN_POLARITY_LOW</i>	channel complementary output polarity is low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config(TIMER0, TIMER_CH_0,
TIMER_OCN_POLARITY_HIGH);
```

### timer\_channel\_output\_state\_config

The description of timer\_channel\_output\_state\_config is shown as below:

**Table 3-974. Function timer\_channel\_output\_state\_config**

<b>Function name</b>	timer_channel_output_state_config
<b>Function prototype</b>	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
<b>Function descriptions</b>	configure TIMER channel enable state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4))
<b>Input parameter{in}</b>	
<b>state</b>	TIMER channel enable state
<i>TIMER_CCX_ENABLE</i>	channel enable
<i>TIMER_CCX_DISABLE</i>	channel disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-975. Function timer\_channel\_complementary\_output\_state\_config**

<b>Function name</b>	timer_channel_complementary_output_state_config
<b>Function prototype</b>	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
<b>Function descriptions</b>	configure TIMER channel complementary output enable state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx(x=0,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx(x=0))
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx(x=0))
<b>Input parameter{in}</b>	
<b>state</b>	TIMER channel complementary output enable state
<i>TIMER_CCXN_ENABLE</i>	channel complementary enable
<i>TIMER_CCXN_DISABLE</i>	channel complementary disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-976. Function timer\_channel\_input\_struct\_para\_init**

<b>Function name</b>	timer_channel_input_struct_para_init
<b>Function prototype</b>	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);

<b>Function descriptions</b>	initialize the parameters of TIMER channel input parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>icpara</b>	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-930. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-977. Function timer\_input\_capture\_config**

<b>Function name</b>	timer_input_capture_config
<b>Function prototype</b>	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
<b>Function descriptions</b>	configure TIMER input capture parameter
<b>Precondition</b>	-
<b>The called functions</b>	timer_channel_input_capture_prescaler_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4))
<b>Input parameter{in}</b>	
<b>icpara</b>	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-930. Structure timer_ic_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* configure TIMER0 input capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
```

### timer\_channel\_input\_capture\_prescaler\_config

The description of timer\_channel\_input\_capture\_prescaler\_config is shown as below:

**Table 3-978. 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
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMERx(x=0..4,15,16))
TIMER_CH_1	TIMER channel 1 (TIMERx(x=0..4))
TIMER_CH_2	TIMER channel 2 (TIMERx(x=0..4))
TIMER_CH_3	TIMER channel 3 (TIMERx(x=0..4))
Input parameter{in}	
prescaler	channel input capture prescaler value
TIMER_IC_PSC_DIV1	no prescaler
TIMER_IC_PSC_DIV2	divided by 2
TIMER_IC_PSC_DIV4	divided by 4
TIMER_IC_PSC_DIV8	divided by 8
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 input capture prescaler value */

timer_channel_input_capture_prescaler_config(TIMER0, TIMER_CH_0,
TIMER_IC_PSC_DIV2);
```

## timer\_channel\_capture\_value\_register\_read

The description of timer\_channel\_capture\_value\_register\_read is shown as below:

**Table 3-979. Function timer\_channel\_capture\_value\_register\_read**

<b>Function name</b>	timer_channel_capture_value_register_read
<b>Function prototype</b>	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
<b>Function descriptions</b>	read TIMER channel capture compare register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4))
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	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 3-980. Function timer\_input\_pwm\_capture\_config**

<b>Function name</b>	timer_input_pwm_capture_config
<b>Function prototype</b>	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);

<b>Function descriptions</b>	configure TIMER input pwm capture function
<b>Precondition</b>	-
<b>The called functions</b>	timer_channel_input_capture_prescaler_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<b>Input parameter{in}</b>	
<b>icpwm</b>	TIMER channel input pwm parameter struct, the structure members can refer to <a href="#">Table 3-930. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure TIMER0 input pwm capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_pwm_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);

```

## timer\_hall\_mode\_config

The description of timer\_hall\_mode\_config is shown as below:

**Table 3-981. Function timer\_hall\_mode\_config**

<b>Function name</b>	timer_hall_mode_config
<b>Function prototype</b>	void timer_hall_mode_config(uint32_t timer_periph, uint32_t hallmode);
<b>Function descriptions</b>	configure TIMER hall sensor mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	

<b>hallmode</b>	TIMER hall sensor mode state
<i>TIMER_HALLINTERFA CE_ENABLE</i>	TIMER hall sensor mode enable
<i>TIMER_HALLINTERFA CE_DISABLE</i>	TIMER hall sensor mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-982. Function timer\_input\_trigger\_source\_select**

<b>Function name</b>	timer_input_trigger_source_select
<b>Function prototype</b>	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);
<b>Function descriptions</b>	select TIMER input trigger source
<b>Precondition</b>	SMC[2:0] = 000
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS EL_ITI0</i>	Internal trigger input 0
<i>TIMER_SMCFG_TRGS EL_ITI1</i>	Internal trigger input 1
<i>TIMER_SMCFG_TRGS EL_ITI2</i>	Internal trigger input 2
<i>TIMER_SMCFG_TRGS EL_ITI3</i>	Internal trigger input 3
<i>TIMER_SMCFG_TRGS EL_CIOF_ED</i>	CIO edge flag
<i>TIMER_SMCFG_TRGS EL_CIOFE0</i>	channel 0 input Filtered output
<i>TIMER_SMCFG_TRGS</i>	channel 1 input Filtered output

<i>EL_CI1FE1</i>	
<i>TIMER_SMCFG_TRGS</i> <i>EL_ETIFP</i>	External trigger input filter output
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-983. Function timer\_master\_output\_trigger\_source\_select**

<b>Function name</b>	timer_master_output_trigger_source_select
<b>Function prototype</b>	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
<b>Function descriptions</b>	select TIMER master mode output trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..5)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>outrigger</b>	master mode control
<i>TIMER_TRI_OUT_SRC</i> <i>_RESET</i>	Reset. When the UPG bit in the <i>TIMERx_SWEVG</i> register is set or a reset is generated by the slave mode controller, a TRGO pulse occurs. And in the latter case, the signal on TRGO is delayed compared to the actual reset
<i>TIMER_TRI_OUT_SRC</i> <i>_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</i> <i>_UPDATE</i>	Update. In this mode the master mode controller selects the update event as TRGO.
<i>TIMER_TRI_OUT_SRC</i> <i>_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</i>	Compare. In this mode the master mode controller selects the O0CPRE

<i>_O0CPRE</i>	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.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-984. Function timer\_slave\_mode\_select**

<b>Function name</b>	timer_slave_mode_select
<b>Function prototype</b>	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
<b>Function descriptions</b>	select TIMER slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>slavemode</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-985. Function timer\_master\_slave\_mode\_config**

<b>Function name</b>	timer_master_slave_mode_config
<b>Function prototype</b>	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);
<b>Function descriptions</b>	configure TIMER master slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>masterslave</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-986. Function timer\_external\_trigger\_config**

<b>Function name</b>	timer_external_trigger_config
<b>Function prototype</b>	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER external trigger input
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	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
<b>Input parameter{in}</b>	
<b>expolarity</b>	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
<b>Input parameter{in}</b>	
<b>extfilter</b>	external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-987. Function timer\_quadrature\_decoder\_mode\_config**

<b>Function name</b>	timer_quadrature_decoder_mode_config
<b>Function prototype</b>	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
<b>Function descriptions</b>	configure TIMER quadrature decoder mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>decomode</b>	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
<b>Input parameter{in}</b>	
<b>ic0polarity</b>	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
<b>Input parameter{in}</b>	
<b>ic1polarity</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-988. Function timer\_internal\_clock\_config**

<b>Function name</b>	timer_internal_clock_config
<b>Function prototype</b>	void timer_internal_clock_config(uint32_t timer_periph);
<b>Function descriptions</b>	configure TIMER internal clock mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-989. Function timer\_internal\_trigger\_as\_external\_clock\_config**

<b>Function name</b>	timer_internal_trigger_as_external_clock_config
<b>Function prototype</b>	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
<b>Function descriptions</b>	configure TIMER the internal trigger as external clock input
<b>Precondition</b>	-
<b>The called functions</b>	timer_input_trigger_source_select
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS EL_ITI0</i>	Internal trigger input 0 (ITI0)
<i>TIMER_SMCFG_TRGS EL_ITI1</i>	Internal trigger input 0 (ITI1)

<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI2</i>	Internal trigger input 0 (ITI2)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI3</i>	Internal trigger input 0 (ITI3)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-990. Function timer\_external\_trigger\_as\_external\_clock\_config**

<b>Function name</b>	timer_external_trigger_as_external_clock_config
<b>Function prototype</b>	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external trigger as external clock input
<b>Precondition</b>	-
<b>The called functions</b>	timer_input_trigger_source_select
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extrigger</b>	external trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CIO edge flag (CIOF_ED)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output (CIOFE0)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI1FE1</i>	channel 1 input Filtered output (CI1FE1)
<b>Input parameter{in}</b>	
<b>expolarity</b>	external trigger polarity
<i>TIMER_IC_POLARITY_</i> <i>RISING</i>	active high or rising edge active
<i>TIMER_IC_POLARITY_</i> <i>FALLING</i>	active low or falling edge active
<i>TIMER_IC_POLARITY_</i> <i>BOTH_EDGE</i>	active both edge

Input parameter{in}	
<b>extfilter</b>	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 3-991. 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}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
Input parameter{in}	
<b>extprescaler</b>	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}	
<b>expolarity</b>	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}	
<b>extfilter</b>	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 3-992. Function timer\_external\_clock\_mode1\_config**

<b>Function name</b>	timer_external_clock_mode1_config
<b>Function prototype</b>	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external clock mode1
<b>Precondition</b>	-
<b>The called functions</b>	timer_external_trigger_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	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
<b>Input parameter{in}</b>	
<b>expolarity</b>	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
<b>Input parameter{in}</b>	
<b>extfilter</b>	ETI external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-993. Function timer\_external\_clock\_mode1\_disable**

<b>Function name</b>	timer_external_clock_mode1_disable
<b>Function prototype</b>	void timer_external_clock_mode1_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER the external clock mode1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 the external clock mode1 */

timer_external_clock_mode1_disable(TIMER0);
```

### timer\_write\_chxval\_register\_config

The description of timer\_write\_chxval\_register\_config is shown as below:

**Table 3-994. Function timer\_write\_chxval\_register\_config**

<b>Function name</b>	timer_write_chxval_register_config
<b>Function prototype</b>	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
<b>Function descriptions</b>	configure TIMER write CHxVAL register selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4, 15, 16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>ccsel</b>	write CHxVAL register selection



<i>TIMER_CHVSEL_DISA</i> <i>BLE</i>	no effect
<i>TIMER_CHVSEL_ENAB</i> <i>LE</i>	when write the CHxVAL register, if the write value is same as the CHxVAL value, the write access is ignored
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-995. Function timer\_output\_value\_selection\_config**

<b>Function name</b>	timer_output_value_selection_config
<b>Function prototype</b>	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);
<b>Function descriptions</b>	configure TIMER output value selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx (x=0,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>outsel</b>	output value selection
<i>TIMER_OUTSEL_DISA</i> <i>BLE</i>	no effect
<i>TIMER_OUTSEL_ENAB</i> <i>LE</i>	if POEN and IOS is 0, the output disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-996. Function timer\_flag\_get**

<b>Function name</b>	timer_flag_get
<b>Function prototype</b>	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
<b>Function descriptions</b>	get TIMER flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag, TIMERx(x=0..5,15,16)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, TIMERx(x=0..4,15,16)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, TIMERx(x=0,15,16)
<i>TIMER_FLAG_TRG</i>	trigger flag, TIMERx(x=0..4)
<i>TIMER_FLAG_BRK</i>	break flag, TIMERx(x=0,15,16)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, TIMERx(x=0..4,15,16)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, TIMERx(x=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TIMERO0 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get(TIMERO0, TIMER_FLAG_UP);
```

## timer\_flag\_clear

The description of timer\_flag\_clear is shown as below:

**Table 3-997. Function timer\_flag\_clear**

<b>Function name</b>	timer_flag_clear
----------------------	------------------

<b>Function prototype</b>	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
<b>Function descriptions</b>	clear TIMER flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag, TIMERx(x=0..5,15,16)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, TIMERx(x=0..4,15,16)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, TIMERx(x=0,15,16)
<i>TIMER_FLAG_TRG</i>	trigger flag, TIMERx(x=0..4)
<i>TIMER_FLAG_BRK</i>	break flag, TIMERx(x=0,15,16)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, TIMERx(x=0..4,15,16)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, TIMERx(x=0..4)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, TIMERx(x=0..4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-998. Function timer\_interrupt\_enable**

<b>Function name</b>	timer_interrupt_enable
<b>Function prototype</b>	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable the TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters

Input parameter{in}	
<b>interrupt</b>	timer interrupt enable source
<i>TIMER_INT_UP</i>	update interrupt enable, TIMERx (x=0..5,15,16)
<i>TIMER_INT_CH0</i>	channel 0 interrupt enable, TIMERx (x=0..4,15,16)
<i>TIMER_INT_CH1</i>	channel 1 interrupt enable, TIMERx (x=0..4)
<i>TIMER_INT_CH2</i>	channel 2 interrupt enable, TIMERx (x=0..4)
<i>TIMER_INT_CH3</i>	channel 3 interrupt enable, TIMERx (x=0..4)
<i>TIMER_INT_CMT</i>	commutation interrupt enable, TIMERx (x=0,15,16)
<i>TIMER_INT_TRG</i>	trigger interrupt enable, TIMERx (x=0..4)
<i>TIMER_INT_BRK</i>	break interrupt enable, TIMERx (x=0,15,16)
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 3-999. Function timer\_interrupt\_disable**

<b>Function name</b>	timer_interrupt_disable
<b>Function prototype</b>	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable the TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>interrupt</b>	timer interrupt disable source
<i>TIMER_INT_UP</i>	update interrupt disable, TIMERx(x=0..5,15,16)
<i>TIMER_INT_CH0</i>	channel 0 interrupt disable, TIMERx(x=0..4,15,16)
<i>TIMER_INT_CH1</i>	channel 1 interrupt disable, TIMERx(x=0..4)
<i>TIMER_INT_CH2</i>	channel 2 interrupt disable, TIMERx(x=0..4)
<i>TIMER_INT_CH3</i>	channel 3 interrupt disable, TIMERx(x=0..4)
<i>TIMER_INT_CMT</i>	commutation interrupt disable, TIMERx(x=0,15,16)
<i>TIMER_INT_TRG</i>	trigger interrupt disable, TIMERx(x=0..4)
<i>TIMER_INT_BRK</i>	break interrupt disable, TIMERx(x=0,15,16)
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 3-1000. Function timer\_interrupt\_flag\_get**

<b>Function name</b>	timer_interrupt_flag_get
<b>Function prototype</b>	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t int_flag);
<b>Function descriptions</b>	get timer interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>int_flag</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..5, 15, 16)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4, 15, 16)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_CMT</i>	channel commutation interrupt flag, <i>TIMERx</i> ( <i>x</i> =0, 15, 16)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, <i>TIMERx</i> ( <i>x</i> =0, 15, 16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-1001. Function timer\_interrupt\_flag\_clear**

<b>Function name</b>	timer_interrupt_flag_clear
<b>Function prototype</b>	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t int_flag);
<b>Function descriptions</b>	clear TIMER interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>int_flag</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..5,15,16)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,15,16)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_CMT</i>	channel commutation interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,15,16)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,15,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER0 update interrupt flag */
```

```
timer_interrupt_flag_clear(TIMER0, TIMER_INT_FLAG_UP);
```

## 3.29. TRNG

The true random number generator (RNG) module can generate a 32-bit value using continuous analog noise. The TRNG registers are listed in chapter [3.29.1](#). the TRNG firmware functions are introduced in chapter [3.29.2](#).

### 3.29.1. Descriptions of Peripheral registers

TRNG registers are listed in the table shown as below:

**Table 3-1002 TRNG Registers**

Registers	Descriptions
TRNG_CTL	TRNG control register
TRNG_STAT	TRNG status register
TRNG_DATA	TRNG data register

## 3.29.2. Descriptions of Peripheral functions

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

**Table 3-1003. TRNG firmware function**

Function name	Function description
trng_deinit	deinitialize the TRNG
trng_enable	enable the TRNG interface
trng_disable	disable the TRNG interface
trng_powermode_config	configure TRNG analog power mode
trng_get_true_random_data	get the true random data
trng_interrupt_enable	enable the TRNG interrupt
trng_interrupt_disable	disable the TRNG interrupt
trng_flag_get	get the TRNG status flags
trng_interrupt_flag_get	get the TRNG interrupt flags
trng_interrupt_flag_clear	clear the TRNG interrupt flags

### Enum trng\_flag\_enum

**Table 3-1004. Enum trng\_flag\_enum**

Member name	Function description
TRNG_FLAG_DRDY	random data ready status
TRNG_FLAG_CECS	clock error current status
TRNG_FLAG_SECS	seed error current status

### Enum trng\_int\_flag\_enum

**Table 3-1005. Enum trng\_int\_flag\_enum**

Member name	Function description
TRNG_INT_FLAG_CEIF	clock error interrupt flag
TRNG_INT_FLAG_SEIF	seed error interrupt flag

### trng\_deinit

The description of trng\_deinit is shown as below:

**Table 3-1006. Function trng\_deinit**

Function name	trng_deinit
Function prototype	void trng_deinit(void)

<b>Function descriptions</b>	reset TRNG peripheral
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset TRNG */
```

```
trng_deinit();
```

### trng\_enable

The description of trng\_enable is shown as below:

**Table 3-1007. Function trng\_enable**

<b>Function name</b>	trng_enable
<b>Function prototype</b>	void trng_enable(void)
<b>Function descriptions</b>	enable the TRNG interface
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TRNG */
```

```
trng_enable();
```

### trng\_disable

The description of trng\_disable is shown as below:

**Table 3-1008 Function trng\_disable**

<b>Function name</b>	trng_disable
<b>Function prototype</b>	void trng_disable(void);
<b>Function descriptions</b>	disable the TRNG interface
<b>Precondition</b>	-



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

Example:

```
/* disable TRNG */
```

```
trng_disable();
```

### trng\_powermode\_config

The description of trng\_powermode\_config is shown as below:

**Table 3-1009. Function trng\_powermode\_config**

Function name	trng_powermode_config
Function prototype	void trng_powermode_config(uint32_t powermode);
Function descriptions	configure TRNG analog power mode
Precondition	trng_conditioning_reset_enable
The called functions	-
Input parameter{in}	
powermode	the power mode selection
TRNG_NR_ULATRL OW	TRNG analog power mode ulatrlow
TRNG_NR_LOW	TRNG analog power mode low
TRNG_NR_MEDIUM	TRNG analog power mode medium
TRNG_NR_HIGH	TRNG analog power mode high
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the TRNG analog power mode as high */
```

```
trng_deinit();
```

```
trng_powermode_config(TRNG_NR_HIGH);
```

```
trng_enable();
```

## trng\_get\_true\_random\_data

The description of trng\_get\_true\_random\_data is shown as below:

**Table 3-1010 Function trng\_get\_true\_random\_data**

<b>Function name</b>	trng_get_true_random_data
<b>Function prototype</b>	uint32_t trng_get_true_random_data(void)
<b>Function descriptions</b>	get the true random data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	the generated random data

Example:

```
/* get true random data */
uint32_t data;
data = trng_get_true_random_data();
```

## trng\_interrupt\_enable

The description of trng\_interrupt\_enable is shown as below:

**Table 3-1011 trng\_interrupt\_enable**

<b>Function name</b>	trng_interrupt_enable
<b>Function prototype</b>	void trng_interrupt_enable(void);
<b>Function descriptions</b>	enable the TRNG interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TRNG interrupt */
trng_interrupt_enable();
```

## trng\_interrupt\_disable

The description of trng\_interrupt\_disable is shown as below:

**Table 3-1012 trng\_interrupt\_disable**

<b>Function name</b>	trng_interrupt_disable
<b>Function prototype</b>	void trng_interrupt_disable(void);
<b>Function descriptions</b>	disable the TRNG interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TRNG interrupt */
trng_interrupt_disable();
```

## trng\_flag\_get

The description of trng\_flag\_get is shown as below:

**Table 3-1013 trng\_flag\_get**

<b>Function name</b>	trng_flag_get
<b>Function prototype</b>	FlagStatus trng_flag_get(trng_flag_enum flag);
<b>Function descriptions</b>	get the trng status flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	trng_flag_enum, please refer to <a href="#">Table 3-1004. Enum trng_flag_enum</a>
TRNG_FLAG_DRDY	random data ready status
TRNG_FLAG_CECS	clock error current status
TRNG_FLAG_SECS	seed error current status
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TRNG clock error current flag status */
FlagStatus flag_status = RESET;
```

```
flag_status == trng_flag_get(TRNG_FLAG_CECS);
```

## trng\_interrupt\_flag\_get

The description of trng\_interrupt\_flag\_get is shown as below:

**Table 3-1014 trng\_interrupt\_flag\_get**

<b>Function name</b>	trng_interrupt_flag_get
<b>Function prototype</b>	FlagStatus trng_interrupt_flag_get(trng_int_flag_enum int_flag);
<b>Function descriptions</b>	get the trng interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	trng_flag_enum, please refer to <a href="#">Table 3-1005. Enum trng_int_flag_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TRNG clock error interrupt flag */
```

```
FlagStatus interrupt_flag = RESET;
```

```
interrupt_flag = trng_interrupt_flag_get(TRNG_INT_FLAG_CEIF);
```

## trng\_interrupt\_flag\_clear

The description of trng\_interrupt\_flag\_clear is shown as below:

**Table 3-1015 trng\_interrupt\_flag\_clear**

<b>Function name</b>	trng_interrupt_flag_clear
<b>Function prototype</b>	void trng_interrupt_flag_get(trng_int_flag_enum int_flag);
<b>Function descriptions</b>	clear the trng interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	trng_flag_enum, please refer to <a href="#">Table 3-1005. Enum trng_int_flag_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TRNG clock error interrupt flag */
```

```
trng_interrupt_flag_clear(TRNG_INT_FLAG_CEIF);
```

### 3.30. TSI

Touch Sensing Interface (TSI) provides a convenient solution for touch keys, sliders and capacitive proximity sensing applications. The TSI registers are listed in chapter [3.30.1](#), the TSI firmware functions are introduced in chapter [3.30.2](#).

#### 3.30.1. Descriptions of Peripheral registers

TSI registers are listed in the table shown as below:

**Table 3-1016. TSI Registers**

Registers	Descriptions
TSI_CTL0	control register0
TSI_INTEN	interrupt enable register
TSI_INTC	interrupt flag clear register
TSI_INTF	interrupt flag register
TSI_PHM	pin hysteresis mode register
TSI_ASW	analog switch register
TSI_SAMPCFG	sample configuration register
TSI_CHCFG	channel configuration register
TSI_GCTL	group control register
TSI_GxCYCN (x=0..2)	group x cycle number registers
TSI_CTL1	control register1

#### 3.30.2. Descriptions of Peripheral functions

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

**Table 3-1017. TSI firmware function**

Function name	Function description
tsi_deinit	reset TSI peripheral
tsi_init	initialize TSI plus prescaler,charge plus,transfer plus,max cycle number
tsi_enable	enable TSI module
tsi_disable	disable TSI module
tsi_sample_pin_enable	enable sample pin
tsi_sample_pin_disable	disable sample pin
tsi_channel_pin_enable	enable channel pin
tsi_channel_pin_disable	disable channel pin
tsi_software_mode_config	configure TSI triggering by software

Function name	Function description
tsi_software_start	start a charge-transfer sequence when TSI is in software trigger mode
tsi_software_stop	stop a charge-transfer sequence when TSI is in software trigger mode
tsi_hardware_mode_config	configure TSI triggering by hardware
tsi_pin_mode_config	configure TSI pin mode when charge-transfer sequence is IDLE
tsi_extend_charge_config	configure extend charge state
tsi_plus_config	configure charge plus and transfer plus
tsi_max_number_config	configure the max cycle number of a charge-transfer sequence
tsi_hysteresis_on	switch on hysteresis pin
tsi_hysteresis_off	switch off hysteresis pin
tsi_analog_on	switch on analog pin
tsi_analog_off	switch off analog pin
tsi_group_enable	enable group
tsi_group_disable	disable group
tsi_group_status_get	get group complete status
tsi_group0_cycle_get	get the cycle number for group0 as soon as a charge-transfer sequence completes
tsi_group1_cycle_get	get the cycle number for group1 as soon as a charge-transfer sequence completes
tsi_group2_cycle_get	get the cycle number for group2 as soon as a charge-transfer sequence completes
tsi_flag_get	get TSI flag
tsi_flag_clear	clear TSI flag
tsi_interrupt_enable	enable TSI interrupt
tsi_interrupt_disable	disable TSI interrupt
tsi_interrupt_flag_get	get TSI interrupt flag
tsi_interrupt_flag_clear	Clear TSI interrupt flag

## tsi\_deinit

The description of tsi\_deinit is shown as below:

**Table 3-1018. Function tsi\_deinit**

Function name	tsi_deinit
Function prototype	void tsi_deinit(void);
Function descriptions	reset TSI peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset TSI*/
```

```
tsi_deinit();
```

## tsi\_init

The description of tsi\_init is shown as below:

**Table 3-1019. Function tsi\_init**

Function name	tsi_init
Function prototype	void tsi_init(uint32_t prescaler,uint32_t charge_duration,uint32_t transfer_duration,uint32_t max_number);
Function descriptions	initialize TSI plus prescaler,charge plus,transfer plus,max cycle number
Precondition	-
The called functions	-
Input parameter{in}	
prescaler	CTCLK clock division factor
TSI_CTCDIV_DIV1	$f_{CTCLK} = f_{HCLK}$
TSI_CTCDIV_DIV2	$f_{CTCLK} = f_{HCLK}/2$
TSI_CTCDIV_DIV4	$f_{CTCLK} = f_{HCLK} /4$
TSI_CTCDIV_DIV8	$f_{CTCLK} = f_{HCLK} /8$
TSI_CTCDIV_DIV16	$f_{CTCLK} = f_{HCLK} /16$
TSI_CTCDIV_DIV32	$f_{CTCLK} = f_{HCLK} /32$
TSI_CTCDIV_DIV64	$f_{CTCLK} = f_{HCLK} /64$
TSI_CTCDIV_DIV128	$f_{CTCLK} = f_{HCLK} /128$
TSI_CTCDIV_DIV256	$f_{CTCLK} = f_{HCLK} /256$
TSI_CTCDIV_DIV512	$f_{CTCLK} = f_{HCLK} /512$
TSI_CTCDIV_DIV1024	$f_{CTCLK} = f_{HCLK} /1024$
TSI_CTCDIV_DIV2048	$f_{CTCLK} = f_{HCLK} /2048$
TSI_CTCDIV_DIV4096	$f_{CTCLK} = f_{HCLK} /4096$
TSI_CTCDIV_DIV8192	$f_{CTCLK} = f_{HCLK} /8192$
TSI_CTCDIV_DIV16384 4	$f_{CTCLK} = f_{HCLK} /16384$
TSI_CTCDIV_DIV32768 8	$f_{CTCLK} = f_{HCLK} /32768$
Input parameter{in}	
charge_duration	charge state duration time
TSI_CHARGE_1CTCL	the duration time of charge state is x CTCLK

$K(x=1..16)$	
<b>Input parameter{in}</b>	
<b>transfer_duration</b>	charge transfer state duration time
<i>TSI_TRANSFER_xCTCLK(x=1..16)SS_CLEAR</i>	the duration time of transfer state is x CTCLK
<b>Input parameter{in}</b>	
<b>max_number</b>	max cycle number
<i>TSI_MAXNUM255</i>	the max cycle number of a sequence is 255
<i>TSI_MAXNUM511</i>	the max cycle number of a sequence is 511
<i>TSI_MAXNUM1023</i>	the max cycle number of a sequence is 1023
<i>TSI_MAXNUM2047</i>	the max cycle number of a sequence is 2047
<i>TSI_MAXNUM4095</i>	the max cycle number of a sequence is 4095
<i>TSI_MAXNUM8191</i>	the max cycle number of a sequence is 8191
<i>TSI_MAXNUM16383</i>	the max cycle number of a sequence is 16383
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* init TSI*/
```

```
tsi_init (TSI_CTCDIV_DIV4096, TSI_CHARGE_10CTCLK, TSI_TRANSFER_8CTCLK,
TSI_MAXNUM511);
```

### tsi\_enable

The description of tsi\_enable is shown as below:

**Table 3-1020. Function tsi\_enable**

<b>Function name</b>	tsi_enable
<b>Function prototype</b>	void tsi_enable (void);
<b>Function descriptions</b>	enable TSI module
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TSI module */
```



```
tsi_enable();
```

## tsi\_disable

The description of tsi\_disable is shown as below:

**Table 3-1021. Function tsi\_disable**

<b>Function name</b>	tsi_disable
<b>Function prototype</b>	void tsi_disable (void);
<b>Function descriptions</b>	disable TSI module
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TSI module */
```

```
tsi_disable ();
```

## tsi\_sample\_pin\_enable

The description of tsi\_sample\_pin\_enable is shown as below:

**Table 3-1022. Function tsi\_sample\_pin\_enable**

<b>Function name</b>	tsi_sample_pin_enable
<b>Function prototype</b>	void tsi_sample_pin_enable(uint32_t sample);
<b>Function descriptions</b>	enable sample pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sample</b>	sample pin
<i>TSI_SAMPCFG_GxPy( x=0..2,y=0..3)</i>	pin y of group x is sample pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable G1P2 sample pin */
```

tsi\_sample\_pin\_enable (TSI\_SAMPCFG\_G1P2);

## tsi\_sample\_pin\_disable

The description of tsi\_sample\_pin\_disable is shown as below:

**Table 3-1023. Function tsi\_sample\_pin\_disable**

<b>Function name</b>	tsi_sample_pin_disable
<b>Function prototype</b>	void tsi_sample_pin_disable(uint32_t sample);
<b>Function descriptions</b>	disable sample pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sample</b>	sample pin
<i>TSI_SAMPCFG_GxPy(</i> <i>x=0..2,y=0..3)</i>	pin y of group x is sample pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable G1P2 sample pin */
```

```
tsi_sample_pin_disable (TSI_SAMPCFG_G1P2);
```

## tsi\_channel\_pin\_enable

The description of tsi\_channel\_pin\_enable is shown as below:

**Table 3-1024. Function tsi\_channel\_pin\_enable**

<b>Function name</b>	tsi_channel_pin_enable
<b>Function prototype</b>	void tsi_channel_pin_enable(uint32_t channel);
<b>Function descriptions</b>	enable channel pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channel</b>	channel pin
<i>TSI_CHCFG_GxPy( x=</i> <i>0..2,y=0..3)</i>	pin y of group x
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable G1P2 channel pin */
```

```
tsi_channel_pin_enable (TSI_CHCFG_G1P2);
```

## tsi\_channel\_pin\_disable

The description of tsi\_channel\_pin\_disable is shown as below:

**Table 3-1025. Function tsi\_channel\_pin\_disable**

<b>Function name</b>	tsi_channel_pin_disable
<b>Function prototype</b>	void tsi_channel_pin_disable(uint32_t channel);
<b>Function descriptions</b>	disable channel pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channel</b>	channel pin
<i>TSI_CHCFG_GxPy</i> (x=0..2,y=0..3)	pin y of group x
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable G1P2 channel pin */
```

```
tsi_channel_pin_disable (TSI_CHCFG_G1P2);
```

## tsi\_software\_mode\_config

The description of tsi\_software\_mode\_config is shown as below:

**Table 3-1026. Function tsi\_software\_mode\_config**

<b>Function name</b>	tsi_software_mode_config
<b>Function prototype</b>	void tsi_software_mode_config (void);
<b>Function descriptions</b>	configure TSI triggering by software
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TSI triggering by software */
```

```
tsi_software_mode_config ();
```

### tsi\_software\_start

The description of tsi\_software\_start is shown as below:

**Table 3-1027. Function tsi\_software\_start**

<b>Function name</b>	tsi_software_start
<b>Function prototype</b>	void tsi_software_start (void);
<b>Function descriptions</b>	start a charge-transfer sequence when TSI is in software trigger mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* start a charge-transfer sequence when TSI is in software trigger mode */
```

```
tsi_software_start ();
```

### tsi\_software\_stop

The description of tsi\_software\_stop is shown as below:

**Table 3-1028. Function tsi\_software\_stop**

<b>Function name</b>	tsi_software_stop
<b>Function prototype</b>	void tsi_software_stop (void);
<b>Function descriptions</b>	stop a charge-transfer sequence when TSI is in software trigger mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* stop a charge-transfer sequence when TSI is in software trigger mode */
```

tsi\_software\_stop ();

## tsi\_hardware\_mode\_config

The description of tsi\_hardware\_mode\_config is shown as below:

**Table 3-1029. Function tsi\_hardware\_mode\_config**

<b>Function name</b>	tsi_hardware_mode_config
<b>Function prototype</b>	void tsi_hardware_mode_config(uint8_t trigger_edge);
<b>Function descriptions</b>	configure TSI triggering by hardware
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>trigger_edge</b>	the edge type in hardware trigger mode
<i>TSI_FALLING_TRIGGER</i>	falling edge trigger TSI charge transfer sequence
<i>TSI_RISING_TRIGGER</i>	rising edge trigger TSI charge transfer sequence
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TSI triggering by hardware */
```

```
tsi_hardware_mode_config (TSI_FALLING_TRIGGER);
```

## tsi\_pin\_mode\_config

The description of tsi\_pin\_mode\_config is shown as below:

**Table 3-1030. Function tsi\_pin\_mode\_config**

<b>Function name</b>	tsi_pin_mode_config
<b>Function prototype</b>	void tsi_pin_mode_config(uint8_t pin_mode);
<b>Function descriptions</b>	configure TSI pin mode when charge-transfer sequence is IDLE
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pin_mode</b>	pin mode when charge-transfer sequence is IDLE
<i>TSI_OUTPUT_LOW</i>	TSI pin will output low when IDLE
<i>TSI_INPUT_FLOATING</i>	TSI pin will keep input_floating when IDLE
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TSI pin mode when charge-transfer sequence is IDLE */
```

```
tsi_pin_mode_config (TSI_OUTPUT_LOW);
```

### tsi\_extend\_charge\_config

The description of tsi\_extend\_charge\_config is shown as below:

**Table 3-1031. Function tsi\_extend\_charge\_config**

<b>Function name</b>	tsi_extend_charge_config
<b>Function prototype</b>	void tsi_extend_charge_config(ControlStatus extend,uint8_t prescaler,uint32_t max_duration);
<b>Function descriptions</b>	configure extend charge state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>extend</b>	enable or disable extend charge state
ENABLE	enable extend charge state
DISABLE	disable extend charge state
<b>Input parameter{in}</b>	
<b>prescaler</b>	ECCLK clock division factor
TSI_EXTEND_DIV1	$f_{ECCLK} = f_{HCLK}$
TSI_EXTEND_DIV2	$f_{ECCLK} = f_{HCLK} / 2$
TSI_EXTEND_DIV3	$f_{ECCLK} = f_{HCLK} / 3$
TSI_EXTEND_DIV4	$f_{ECCLK} = f_{HCLK} / 4$
TSI_EXTEND_DIV5	$f_{ECCLK} = f_{HCLK} / 5$
TSI_EXTEND_DIV6	$f_{ECCLK} = f_{HCLK} / 6$
TSI_EXTEND_DIV7	$f_{ECCLK} = f_{HCLK} / 7$
TSI_EXTEND_DIV8	$f_{ECCLK} = f_{HCLK} / 8$
<b>Input parameter{in}</b>	
<b>max_duration</b>	extend charge state maximum duration time
value range 1...128	extend charge state maximum duration time is $1 \cdot t_{ECCLK} \sim 128 \cdot t_{ECCLK}$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure extend charge state */
```

```
tsi_extend_charge_config (ENABLE, TSI_EXTEND_DIV2, 10);
```

## tsi\_plus\_config

The description of tsi\_plus\_config is shown as below:

**Table 3-1032. Function tsi\_plus\_config**

<b>Function name</b>	tsi_plus_config
<b>Function prototype</b>	void tsi_plus_config(uint32_t prescaler,uint32_t charge_duration,uint32_t transfer_duration);
<b>Function descriptions</b>	configure charge plus and transfer plus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>prescaler</b>	CTCLK clock division factor
<i>TSI_CTCDIV_DIV1</i>	$f_{CTCLK} = f_{HCLK}$
<i>TSI_CTCDIV_DIV2</i>	$f_{CTCLK} = f_{HCLK} / 2$
<i>TSI_CTCDIV_DIV4</i>	$f_{CTCLK} = f_{HCLK} / 4$
<i>TSI_CTCDIV_DIV8</i>	$f_{CTCLK} = f_{HCLK} / 8$
<i>TSI_CTCDIV_DIV16</i>	$f_{CTCLK} = f_{HCLK} / 16$
<i>TSI_CTCDIV_DIV32</i>	$f_{CTCLK} = f_{HCLK} / 32$
<i>TSI_CTCDIV_DIV64</i>	$f_{CTCLK} = f_{HCLK} / 64$
<i>TSI_CTCDIV_DIV128</i>	$f_{CTCLK} = f_{HCLK} / 128$
<i>TSI_CTCDIV_DIV256</i>	$f_{CTCLK} = f_{HCLK} / 256$
<i>TSI_CTCDIV_DIV512</i>	$f_{CTCLK} = f_{HCLK} / 512$
<i>TSI_CTCDIV_DIV1024</i>	$f_{CTCLK} = f_{HCLK} / 1024$
<i>TSI_CTCDIV_DIV2048</i>	$f_{CTCLK} = f_{HCLK} / 2048$
<i>TSI_CTCDIV_DIV4096</i>	$f_{CTCLK} = f_{HCLK} / 4096$
<i>TSI_CTCDIV_DIV8192</i>	$f_{CTCLK} = f_{HCLK} / 8192$
<i>TSI_CTCDIV_DIV16384</i> 4	$f_{CTCLK} = f_{HCLK} / 16384$
<i>TSI_CTCDIV_DIV32768</i> 8	$f_{CTCLK} = f_{HCLK} / 32768$
<b>Input parameter{in}</b>	
<b>charge_duration</b>	charge state duration time
<i>TSI_CHARGE_1CTCL</i> $K(x=1..16)$	the duration time of charge state is x CTCLK
<b>Input parameter{in}</b>	
<b>transfer_duration</b>	charge transfer state duration time
<i>TSI_TRANSFER_xCTC</i> $LK(x=1..16)$	the duration time of transfer state is x CTCLK
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure charge plus and transfer plus */
```

```
tsi_plus_config (TSI_CTCDIV_DIV1024, TSI_CHARGE_10CTCLK,  
TSI_TRANSFER_8CTCLK);
```

### tsi\_max\_number\_config

The description of tsi\_max\_number\_config is shown as below:

**Table 3-1033. Function tsi\_max\_number\_config**

Function name	tsi_max_number_config
Function prototype	void tsi_max_number_config(uint32_t max_number);
Function descriptions	configure the max cycle number of a charge-transfer sequence
Precondition	-
The called functions	-
Input parameter{in}	
max_number	max cycle number
TSI_MAXNUM255	the max cycle number of a sequence is 255
TSI_MAXNUM511	the max cycle number of a sequence is 511
TSI_MAXNUM1023	the max cycle number of a sequence is 1023
TSI_MAXNUM2047	the max cycle number of a sequence is 2047
TSI_MAXNUM4095	the max cycle number of a sequence is 4095
TSI_MAXNUM8191	the max cycle number of a sequence is 8191
TSI_MAXNUM16383	the max cycle number of a sequence is 16383
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the max cycle number of a charge-transfer sequence */
```

```
tsi_max_number_config (TSI_MAXNUM1023);
```

### tsi\_hysteresis\_on

The description of tsi\_hysteresis\_on is shown as below:

**Table 3-1034. Function tsi\_hysteresis\_on**

Function name	tsi_hysteresis_on
Function prototype	void tsi_hysteresis_on(uint32_t group_pin);
Function descriptions	switch on hysteresis pin
Precondition	-
The called functions	-



Input parameter{in}	
<b>group_pin</b>	select pin which will be switched on hysteresis
<i>TSI_PHM_GxPy</i> (x=0..2, y=0..3)	pin y of group x switch on hysteresis
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* switch on hysteresis pin */
```

```
tsi_hysteresis_on (TSI_PHM_G1P2);
```

### tsi\_hysteresis\_off

The description of tsi\_hysteresis\_off is shown as below:

**Table 3-1035. Function tsi\_hysteresis\_off**

<b>Function name</b>	tsi_hysteresis_off
<b>Function prototype</b>	void tsi_hysteresis_off(uint32_t group_pin);
<b>Function descriptions</b>	switch off hysteresis pin
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>group_pin</b>	select pin which will be switched off hysteresis
<i>TSI_PHM_GxPy</i> (x=0..2, y=0..3)	pin y of group x switch off hysteresis
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* switch off hysteresis pin */
```

```
tsi_hysteresis_off (TSI_PHM_G1P2);
```

### tsi\_analog\_on

The description of tsi\_analog\_on is shown as below:

**Table 3-1036. Function tsi\_analog\_on**

<b>Function name</b>	tsi_analog_on
<b>Function prototype</b>	void tsi_analog_on(uint32_t group_pin);
<b>Function descriptions</b>	switch on analog pin

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>group_pin</b>	select pin which will be switched on analog
<i>TSI_ASW_GxPy</i> (x=0..2,y=0..3)	pin y of group x switch on analog
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* switch on analog pin */
```

```
tsi_analog_on (TSI_ASW_G1P2);
```

## tsi\_analog\_off

The description of tsi\_analog\_off is shown as below:

**Table 3-1037. Function tsi\_analog\_off**

<b>Function name</b>	tsi_analog_off
<b>Function prototype</b>	void tsi_analog_off(uint32_t group_pin);
<b>Function descriptions</b>	switch off analog pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>group_pin</b>	select pin which will be switched off analog
<i>TSI_ASW_GxPy</i> (x=0..2,y=0..3)	pin y of group x switch off analog
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* switch off analog pin */
```

```
tsi_analog_off (TSI_ASW_G1P2);
```

## tsi\_group\_enable

The description of tsi\_group\_enable is shown as below:

**Table 3-1038. Function tsi\_group\_enable**

<b>Function name</b>	tsi_group_enable
----------------------	------------------

<b>Function prototype</b>	void tsi_group_enable(uint32_t group);
<b>Function descriptions</b>	enable group
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>group</b>	select group to be enabled
<i>TSI_GCTL_GEx(x=0..2)</i>	the x group will be enabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable group 2 */
```

```
tsi_group_enable (TSI_GCTL_GE2);
```

### tsi\_group\_disable

The description of tsi\_group\_disable is shown as below:

**Table 3-1039. Function tsi\_group\_disable**

<b>Function name</b>	tsi_group_disable
<b>Function prototype</b>	void tsi_group_disable(uint32_t group);
<b>Function descriptions</b>	disable group
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>group</b>	select group to be disabled
<i>TSI_GCTL_GEx(x=0..2)</i>	the x group will be disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable group 2 */
```

```
tsi_group_disable (TSI_GCTL_GE2);
```

### tsi\_group\_status\_get

The description of tsi\_group\_status\_get is shown as below:

**Table 3-1040. Function tsi\_group\_status\_get**

<b>Function name</b>	tsi_group_status_get
----------------------	----------------------

<b>Function prototype</b>	FlagStatus tsi_group_status_get(uint32_t group);
<b>Function descriptions</b>	get group complete status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>group</b>	select group
TSI_GCTL_GCx(x=0..2)	get the complete status of group x
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get group complete status */
```

```
FlagStatus flag = RESET;
```

```
flag = tsi_flag_get (TSI_GCTL_GC2);
```

### tsi\_group0\_cycle\_get

The description of tsi\_group0\_cycle\_get is shown as below:

**Table 3-1041. Function tsi\_group0\_cycle\_get**

<b>Function name</b>	tsi_group0_cycle_get
<b>Function prototype</b>	uint16_t tsi_group0_cycle_get(void);
<b>Function descriptions</b>	get the cycle number for group0 as soon as a charge-transfer sequence completes
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	0-8192

Example:

```
/* get the cycle number for group0 as soon as a charge-transfer sequence completes */
```

```
uint16_t flag = 0;
```

```
flag = tsi_group0_cycle_get ();
```

## tsi\_group1\_cycle\_get

The description of tsi\_group1\_cycle\_get is shown as below:

**Table 3-1042. Function tsi\_group1\_cycle\_get**

<b>Function name</b>	tsi_group1_cycle_get
<b>Function prototype</b>	uint16_t tsi_group1_cycle_get(void);
<b>Function descriptions</b>	get the cycle number for group1 as soon as a charge-transfer sequence completes
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	0-8192

Example:

```
/* get the cycle number for group1 as soon as a charge-transfer sequence completes */
uint16_t flag = 0;

flag = tsi_group1_cycle_get ();
```

## tsi\_group2\_cycle\_get

The description of tsi\_group2\_cycle\_get is shown as below:

**Table 3-1043. Function tsi\_group2\_cycle\_get**

<b>Function name</b>	tsi_group2_cycle_get
<b>Function prototype</b>	uint16_t tsi_group2_cycle_get(void);
<b>Function descriptions</b>	get the cycle number for group2 as soon as a charge-transfer sequence completes
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	0-8192

Example:

```
/* get the cycle number for group2 as soon as a charge-transfer sequence completes */
```

```
uint16_t flag = 0;
```

```
flag = tsi_group2_cycle_get ();
```

## tsi\_flag\_clear

The description of tsi\_flag\_clear is shown as below:

**Table 3-1044. Function tsi\_flag\_clear**

<b>Function name</b>	tsi_flag_clear
<b>Function prototype</b>	void tsi_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear TSI flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	select flag which will be cleared
<i>TSI_FLAG_CTCF</i>	clear charge-transfer complete flag
<i>TSI_FLAG_MNERR</i>	clear max cycle number error
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TSI_FLAG_CTCF_CLR flag */
```

```
tsi_flag_clear (TSI_FLAG_CTCF_CLR);
```

## tsi\_flag\_get

The description of tsi\_flag\_get is shown as below:

**Table 3-1045. Function tsi\_flag\_get**

<b>Function name</b>	tsi_flag_get
<b>Function prototype</b>	FlagStatus tsi_flag_get(uint32_t flag);
<b>Function descriptions</b>	get TSI flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag
<i>TSI_FLAG_CTCF</i>	charge-transfer complete flag
<i>TSI_FLAG_MNERR</i>	max Cycle Number Error
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TSI_FLAG_CTCF flag */
```

```
FlagStatus flag = RESET;
```

```
flag = tsi_flag_get (TSI_FLAG_CTCF);
```

### tsi\_interrupt\_enable

The description of tsi\_interrupt\_enable is shown as below:

**Table 3-1046. Function tsi\_interrupt\_enable**

<b>Function name</b>	tsi_interrupt_enable
<b>Function prototype</b>	void tsi_interrupt_enable(uint32_t source);
<b>Function descriptions</b>	enable TSI interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	select interrupt which will be enabled
<i>TSI_INT_CCTCF</i>	charge-transfer complete flag interrupt enable
<i>TSI_INT_MNERR</i>	max cycle number error interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TSI_INT_CCTCF interrupt */
```

```
tsi_interrupt_enable (TSI_INT_CCTCF);
```

### tsi\_interrupt\_disable

The description of tsi\_interrupt\_disable is shown as below:

**Table 3-1047. Function tsi\_interrupt\_disable**

<b>Function name</b>	tsi_interrupt_disable
<b>Function prototype</b>	void tsi_interrupt_disable(uint32_t source);
<b>Function descriptions</b>	disable TSI interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	select interrupt which will be disabled
<i>TSI_INT_CCTCF</i>	charge-transfer complete flag interrupt disable
<i>TSI_INT_MNERR</i>	max cycle number error interrupt disable
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* disable TSI_INT_CCTCF interrupt */
tsi_interrupt_disable (TSI_INT_CCTCF);
```

### tsi\_interrupt\_flag\_clear

The description of tsi\_interrupt\_flag\_clear is shown as below:

**Table 3-1048. Function tsi\_interrupt\_flag\_clear**

<b>Function name</b>	tsi_interrupt_flag_clear
<b>Function prototype</b>	void tsi_interrupt_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear TSI interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	select flag which will be cleared
<i>TSI_INT_FLAG_CTCF</i>	clear charge-transfer complete flag
<i>TSI_INT_FLAG_MNER</i> <i>R</i>	clear max cycle number error
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TSI_INT_FLAG_CTCF interrupt flag */
tsi_interrupt_flag_clear (TSI_INT_FLAG_CTCF);
```

### tsi\_interrupt\_flag\_get

The description of tsi\_interrupt\_flag\_get is shown as below:

**Table 3-1049. Function tsi\_interrupt\_flag\_get**

<b>Function name</b>	tsi_interrupt_flag_get
<b>Function prototype</b>	FlagStatus tsi_interrupt_flag_get(uint32_t flag);
<b>Function descriptions</b>	get TSI interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag



<i>TSI_INT_FLAG_CTCF</i>	charge-transfer complete flag
<i>TSI_INT_FLAG_MNER</i> <i>R</i>	max Cycle Number Error
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TSI_INT_FLAG_CTCF interrupt flag */
```

```
FlagStatus flag = RESET;
```

```
flag = tsi_interrupt_flag_get (TSI_INT_FLAG_CTCF);
```

## 3.31. TZPCU

TZPCU is used to implement TrustZone protection functions for the devices. The TZPCU registers are listed in chapter [3.31.1](#), the TZPCU firmware functions are introduced in chapter [3.31.2](#).

### 3.31.1. Descriptions of Peripheral registers

TZPCU registers are listed in the table shown as below:

**Table 3-1050. TZPCU Registers**

Registers	Descriptions
TZSPC registers	
TZPCU_TZSPC_CTL	TZSPC control register
TZPCU_TZSPC_SAM_CFG0	TZSPC secure access mode configuration register 0
TZPCU_TZSPC_SAM_CFG1	TZSPC secure access mode configuration register 1
TZPCU_TZSPC_SAM_CFG2	TZSPC secure access mode configuration register 2
TZPCU_TZSPC_PAM_CFG0	TZSPC privilege access mode configuration register 0
TZPCU_TZSPC_PAM_CFG1	TZSPC privilege access mode configuration register 1
TZPCU_TZSPC_PAM_CFG2	TZSPC privilege access mode configuration register 2
TZPCU_TZSPC_TZMMPC0_NSM0	TZSPC external memory 0 non-secure mark register 0
TZPCU_TZSPC_TZMMPC0_NSM1	TZSPC external memory 0 non-secure mark register 1
TZPCU_TZSPC_TZMMPC0_NSM2	TZSPC external memory 0 non-secure mark register 2
TZPCU_TZSPC_TZMMPC0_NSM3	TZSPC external memory 0 non-secure mark register 3
TZPCU_TZSPC_TZMMPC1_NSM0	TZSPC external memory 1 non-secure mark register 0
TZPCU_TZSPC_TZMMPC1_NSM1	TZSPC external memory 1 non-secure mark register 1
TZPCU_TZSPC_DBG_CFG	TZSPC debug configuration register
TZBMPC registers	
TZPCU_TZBMPC_CTL	TZBMPC control register

Registers	Descriptions
TZPCU_TZBMPC_VEC	TZBMPC vector register
TZPCU_TZBMPC_LOCK0	TZBMPC lock register 0
TZIAC registers	
TZPCU_TZIAC_INTEN0	TZIAC enable register 0
TZPCU_TZIAC_INTEN1	TZIAC enable register 1
TZPCU_TZIAC_INTEN2	TZIAC enable register 2
TZPCU_TZIAC_STAT0	TZIAC status register 0
TZPCU_TZIAC_STAT1	TZIAC status register 1
TZPCU_TZIAC_STAT2	TZIAC status register 2
TZPCU_TZIAC_STATC0	TZIAC flag clear register 0
TZPCU_TZIAC_STATC1	TZIAC flag clear register 1
TZPCU_TZIAC_STATC2	TZIAC flag clear register 2

## 3.31.2. Descriptions of Peripheral functions

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

**Table 3-1051. TZPCU firmware function**

Function name	Function description
TZSPC function	
tzpcu_tzspc_peripheral_attributes_config	configure peripherals secure attributes
tzpcu_tzspc_peripheral_attributes_get	get peripherals secure attributes
tzpcu_non_secure_mark_struct_parameter_init	initialize the TZPCU non-secure mark struct with the default values
tzpcu_tzspc_emnsm_config	configure external memory non-secure mark
tzpcu_tzspc_items_lock	lock TZSPC items
tzpcu_tzspc_dbg_config	configure debug type
TZBMPC function	
tzpcu_tzbmpc_lock	lock the control register of the TZBMPC sub-block
tzpcu_tzbmpc_security_state_config	configure default security state
tzpcu_tzbmpc_secure_access_config	configure secure access state
tzpcu_tzbmpc_block_secure_access_mode_config	configure block secure access mode
tzpcu_tzbmpc_union_block_lock	lock configure union block secure access mode
TZIAC function	
tzpcu_tziac_interrupt_enable	enable illegal access interrupt
tzpcu_tziac_interrupt_disable	disable illegal access interrupt
tzpcu_tziac_flag_get	get illegal access interrupt flag
tzpcu_tziac_flag_clear	clear illegal access interrupt flag

## Enum tzpcu\_mem

**Table 3-1052. tzpcu\_mem**

enum name	enum description
QSPI_FLASH_MEM	QSPI flash memory
SQPI_PSRAM_MEM	SQPI PSRAM memory

## Enum tzpcu\_non\_secure\_mark\_region

**Table 3-1053. tzpcu\_non\_secure\_mark\_region**

enum name	enum description
NON_SECURE_MARK_REGION0	non-secure mark region 0
NON_SECURE_MARK_REGION1	non-secure mark region 1
NON_SECURE_MARK_REGION2	non-secure mark region 2 (QSPI flash only)
NON_SECURE_MARK_REGION3	non-secure mark region 3 (QSPI flash only)

## Structure tzpcu\_non\_secure\_mark\_struct

**Table 3-1054. tzpcu\_non\_secure\_mark\_struct**

Member name	Function description
memory_type	memory type, refer to <a href="#">Table 3-1052. tzpcu_mem</a>
region_number	non secure mark region number, refer to <a href="#">Table 3-1053. tzpcu_non_secure_mark_region</a>
start_address	external memory non-secure area start address
length	external memory non-secure area length

## tzpcu\_tzspc\_peripheral\_attributes\_config

The description of tzpcu\_tzspc\_peripheral\_attributes\_config is shown as below:

**Table 3-1055. Function tzpcu\_tzspc\_peripheral\_attributes\_config**

<b>Function name</b>	tzpcu_tzspc_peripheral_attributes_config
<b>Function prototype</b>	void tzpcu_tzspc_peripheral_attributes_config(uint32_t periph, uint32_t attributes);
<b>Function descriptions</b>	configure peripherals secure attributes
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	peripheral
<i>TZPCU_PERIPH_TIME</i> <i>R1</i>	TIMER1 peripheral
<i>TZPCU_PERIPH_TIME</i> <i>R2</i>	TIMER2 peripheral
<i>TZPCU_PERIPH_TIME</i> <i>R3</i>	TIMER3 peripheral

TZPCU_PERIPH_TIME R4	TIMER4 peripheral
TZPCU_PERIPH_TIME R5	TIMER5 peripheral
TZPCU_PERIPH_WW DGT	WWDGT peripheral
TZPCU_PERIPH_FWD GT	FWDGT peripheral
TZPCU_PERIPH_SPI1	SPI1 peripheral
TZPCU_PERIPH_USA RT1	USART1 peripheral
TZPCU_PERIPH_USA RT2	USART2 peripheral
TZPCU_PERIPH_I2C0	I2C0 peripheral
TZPCU_PERIPH_I2C1	I2C1 peripheral
TZPCU_PERIPH_USB FS	USBFS peripheral
TZPCU_PERIPH_TIME R0	TIMER0 peripheral
TZPCU_PERIPH_SPI0	SPI0 peripheral
TZPCU_PERIPH_USA RT0	USART0 peripheral
TZPCU_PERIPH_TIME R15	TIMER15 peripheral
TZPCU_PERIPH_TIME R16	TIMER16 peripheral
TZPCU_PERIPH_HPD F	HPDF peripheral(HPDF not support on GD32W515Tx series devices)
TZPCU_PERIPH_CRC	CRC peripheral
TZPCU_PERIPH_TSI	TSI peripheral
TZPCU_PERIPH_ICAC HE	ICACHE peripheral
TZPCU_PERIPH_ADC	ADC peripheral
TZPCU_PERIPH_CAU	CAU peripheral
TZPCU_PERIPH_HAU	HAU peripheral
TZPCU_PERIPH_TRN G	TRNG peripheral
TZPCU_PERIPH_PKC AU	PKCAU peripheral
TZPCU_PERIPH_SDIO	SDIO peripheral
TZPCU_PERIPH_EFU SE	EFUSE peripheral
TZPCU_PERIPH_DBG	DBG peripheral(only for privilege attributes)

<code>TZPCU_PERIPH_SQPI</code> <code>_PSRAMREG</code>	SQPI PSRAMREG peripheral
<code>TZPCU_PERIPH_QSPI</code> <code>_FLASHREG</code>	QSPI FLASHREG peripheral
<code>TZPCU_PERIPH_WIFI</code> <code>_RF</code>	WIFI RF peripheral
<code>TZPCU_PERIPH_I2S1</code> <code>_ADD</code>	I2S1_ADD peripheral
<code>TZPCU_PERIPH_DCI</code>	DCI peripheral(DCI not support on GD32W515Tx series devices)
<code>TZPCU_PERIPH_WIFI</code>	WIFI peripheral
<code>TZPCU_PERIPH_ALL</code>	all peripherals
<b>Input parameter{in}</b>	
<b>attributes</b>	security and privilege attributes
<code>TZPCU_SEC</code>	peripheral secure attributes is secure, exclusive with <code>TZPCU_NSEC</code>
<code>TZPCU_NSEC</code>	peripheral secure attributes is non-secure, exclusive with <code>TZPCU_SEC</code>
<code>TZPCU_PRIV</code>	peripheral privilege attributes is privilege, exclusive with <code>TZPCU_NPRIV</code>
<code>TZPCU_NPRIV</code>	peripheral privilege attributes is non-privilege, exclusive with <code>TZPCU_PRIV</code>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER1 secure attributes to secure and non-privilege */
```

```
tzpcu_tzspc_peripheral_attributes_config(TZPCU_PERIPH_TIMER1, TZPCU_SEC |  
TZPCU_NPRIV);
```

## tzpcu\_tzspc\_peripheral\_attributes\_get

The description of `tzpcu_tzspc_peripheral_attributes_get` is shown as below:

**Table 3-1056. Function `tzpcu_tzspc_peripheral_attributes_get`**

<b>Function name</b>	<code>tzpcu_tzspc_peripheral_attributes_get</code>
<b>Function prototype</b>	<code>uint32_t tzpcu_tzspc_peripheral_attributes_get(uint32_t periph);</code>
<b>Function descriptions</b>	get peripherals secure attributes
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	peripheral
<code>TZPCU_PERIPH_TIMER1</code> <i>R1</i>	TIMER1 peripheral
<code>TZPCU_PERIPH_TIMER2</code> <i>R2</i>	TIMER2 peripheral

TZPCU_PERIPH_TIME R3	TIMER3 peripheral
TZPCU_PERIPH_TIME R4	TIMER4 peripheral
TZPCU_PERIPH_TIME R5	TIMER5 peripheral
TZPCU_PERIPH_WW DGT	WWDGT peripheral
TZPCU_PERIPH_FWD GT	FWDGT peripheral
TZPCU_PERIPH_SPI1	SPI1 peripheral
TZPCU_PERIPH_USA RT1	USART1 peripheral
TZPCU_PERIPH_USA RT2	USART2 peripheral
TZPCU_PERIPH_I2C0	I2C0 peripheral
TZPCU_PERIPH_I2C1	I2C1 peripheral
TZPCU_PERIPH_USB FS	USBFS peripheral
TZPCU_PERIPH_TIME R0	TIMER0 peripheral
TZPCU_PERIPH_SPI0	SPI0 peripheral
TZPCU_PERIPH_USA RT0	USART0 peripheral
TZPCU_PERIPH_TIME R15	TIMER15 peripheral
TZPCU_PERIPH_TIME R16	TIMER16 peripheral
TZPCU_PERIPH_HPD F	HPDF peripheral(c)
TZPCU_PERIPH_CRC	CRC peripheral
TZPCU_PERIPH_TSI	TSI peripheral
TZPCU_PERIPH_ICAC HE	ICACHE peripheral
TZPCU_PERIPH_ADC	ADC peripheral
TZPCU_PERIPH_CAU	CAU peripheral
TZPCU_PERIPH_HAU	HAU peripheral
TZPCU_PERIPH_TRN G	TRNG peripheral
TZPCU_PERIPH_PKC AU	PKCAU peripheral
TZPCU_PERIPH_SDIO	SDIO peripheral
TZPCU_PERIPH_EFU	EFUSE peripheral

SE	
TZPCU_PERIPH_DBG	DBG peripheral(only for privilege attributes)
TZPCU_PERIPH_QSPI _PSRAMREG	SQPI PSRAMREG peripheral
TZPCU_PERIPH_QSPI _FLASHREG	QSPI FLASHREG peripheral
TZPCU_PERIPH_WIFI _RF	WIFI RF peripheral
TZPCU_PERIPH_I2S1 _ADD	I2S1_ADD peripheral
TZPCU_PERIPH_DCI	DCI peripheral(DCI not support on GD32W515Tx series devices)
TZPCU_PERIPH_WIFI	WIFI peripheral
TZPCU_PERIPH_ALL	all peripherals
Output parameter{out}	
-	-
Return value	
uint32_t	TZPCU_SEC/TZPCU_NSEC or TZPCU_PRIV/TZPCU_NPRIV

Example:

```
/* get TIMER1 secure attributes */
```

```
uint32_t sec_attr = tzpcu_tzspc_peripheral_attributes_get(TZPCU_PERIPH_TIMER1);
```

## tzpcu\_non\_secure\_mark\_struct\_para\_init

The description of tzpcu\_non\_secure\_mark\_struct\_para\_init is shown as below:

**Table 3-1057. Function tzpcu\_non\_secure\_mark\_struct\_para\_init**

Function name	tzpcu_non_secure_mark_struct_para_init
Function prototype	void tzpcu_non_secure_mark_struct_para_init(tzpcu_non_secure_mark_struct* init_struct);
Function descriptions	initialize the TZPCU non-secure mark struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
init_struct	TZPCU non-secure mark init parameter struct, the structure members can refer to <a href="#">Table 3-1054. tzpcu_non_secure_mark_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the TZPCU non-secure mark struct with the default values */
```

```
tzpcu_non_secure_mark_struct init_struct;
```

```
tzpcu_non_secure_mark_struct_para_init(&init_struct);
```

### tzpcu\_tzspc\_emnsm\_config

The description of tzpcu\_tzspc\_emnsm\_config is shown as below:

**Table 3-1058. Function tzpcu\_tzspc\_emnsm\_config**

<b>Function name</b>	tzpcu_tzspc_emnsm_config
<b>Function prototype</b>	ErrStatus tzpcu_tzspc_emnsm_config(tzpcu_non_secure_mark_struct *p_non_secure_mark);
<b>Function descriptions</b>	configure external memory non-secure mark
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>p_non_secure_mark</b>	TZPCU non-secure mark struct, the structure members can refer to <a href="#">Table 3-1054. tzpcu_non_secure_mark_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* configure external memory QSPI FLASH first 8KB space to non-secure */
```

```
tzpcu_non_secure_mark_struct non_secure_mark;
```

```
ErrStatus status = ERROR;
```

```
tzpcu_non_secure_mark_struct_para_init(&non_secure_mark);
```

```
non_secure_mark.memory_type = QSPI_FLASH_MEM;
```

```
non_secure_mark.region_number = NON_SECURE_MARK_REGION0;
```

```
non_secure_mark.start_address = 0U;
```

```
non_secure_mark.length = 0x00002000;
```

```
status = tzpcu_tzspc_emnsm_config(&non_secure_mark);
```

### tzpcu\_tzspc\_items\_lock

The description of tzpcu\_tzspc\_items\_lock is shown as below:

**Table 3-1059. Function tzpcu\_tzspc\_items\_lock**

<b>Function name</b>	tzpcu_tzspc_items_lock
----------------------	------------------------



<b>Function prototype</b>	void tzpcu_tzspc_items_lock(void);
<b>Function descriptions</b>	lock tzspc items
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock tzspc items */
tzpcu_tzspc_items_lock();
```

## tzpcu\_tzspc\_dbg\_config

The description of tzpcu\_tzspc\_dbg\_config is shown as below:

**Table 3-1060. Function tzpcu\_tzspc\_dbg\_config**

<b>Function name</b>	tzpcu_tzspc_dbg_config
<b>Function prototype</b>	void tzpcu_tzspc_dbg_config(uint32_t dbg_type, ControlStatus config_value);
<b>Function descriptions</b>	configure debug type
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dbg_type</b>	debug type
<i>INVASIVE_DEBUG</i>	invasive debug
<i>NON_INVASIVE_DEBUG</i>	non-invasive debug
<i>SECURE_INVASIVE_DEBUG</i>	secure invasive debug
<i>SECURE_NON_INVASIVE_DEBUG</i>	secure non-Invasive debug
<b>Input parameter{in}</b>	
<b>config_value</b>	ENABLE or DISABLE
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable invasive debug */
```

```
tzpcu_tzspc_dbg_config(INVASIVE_DEBUG, DISABLE);
```

### tzpcu\_tzbmpc\_lock

The description of tzpcu\_tzbmpc\_lock is shown as below:

**Table 3-1061. Function tzpcu\_tzbmpc\_lock**

<b>Function name</b>	tzpcu_tzbmpc_lock
<b>Function prototype</b>	void tzpcu_tzbmpc_lock(uint32_t tzbmpx);
<b>Function descriptions</b>	lock the control register of the TZBMPC sub-block
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>tzbmp</b>	TZBMPC
<i>TZBMPCx</i>	TZBMPCx(x=0,1,2,3)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock the control register of the TZBMPC0 sub-block */
```

```
tzpcu_tzbmpc_lock(TZBMPC0);
```

### tzpcu\_tzbmpc\_security\_state\_config

The description of tzpcu\_tzbmpc\_security\_state\_config is shown as below:

**Table 3-1062. Function tzpcu\_tzbmpc\_security\_state\_config**

<b>Function name</b>	tzpcu_tzbmpc_security_state_config
<b>Function prototype</b>	void tzpcu_tzbmpc_security_state_config(uint32_t tzbmpx, uint32_t sec_state);
<b>Function descriptions</b>	configure default security state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>tzbmp</b>	TZBMPC
<i>TZBMPCx</i>	TZBMPCx(x=0,1,2,3)
<b>Input parameter{in}</b>	
<b>sec_state</b>	security state of TZBMPC source clock
<i>DEFAULT_STATE</i>	default state, if do not exist secure area, source clock is non-secure
<i>INVERT_STATE</i>	invert the state, even if do not exist secure area, source clock is still secure
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* configure TZBMPC0 default security state to invert state */
```

```
tzpcu_tzbmpc_security_state_config(TZBMPC0, INVERT_STATE);
```

## **tzpcu\_tzbmpc\_secure\_access\_config**

The description of tzpcu\_tzbmpc\_secure\_access\_config is shown as below:

**Table 3-1063. Function tzpcu\_tzbmpc\_secure\_access\_config**

<b>Function name</b>	tzpcu_tzbmpc_secure_access_config
<b>Function prototype</b>	void tzpcu_tzbmpc_secure_access_config(uint32_t tzbmpx, uint32_t sec_ilaccess_state);
<b>Function descriptions</b>	configure secure access state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>tzbmp</b>	TZBMPC
<i>TZBMPCx</i>	TZBMPCx(x=0,1,2,3)
<b>Input parameter{in}</b>	
<b>sec_ilaccess_state</b>	secure illegal access state
<i>SECURE_ILLEGAL_ACCESS_ENABLE</i>	secure read/write access non-secure SRAM is illegal
<i>SECURE_ILLEGAL_ACCESS_DISABLE</i>	secure read/write access non-secure SRAM is legal
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TZBMPC0 secure access state to be enabled , secure read/write access non-secure SRAM is illegal */
```

```
tzpcu_tzbmpc_secure_access_config(TZBMPC0, SECURE_ILLEGAL_ACCESS_ENABLE);
```

## **tzpcu\_tzbmpc\_block\_secure\_access\_mode\_config**

The description of tzpcu\_tzbmpc\_block\_secure\_access\_mode\_config is shown as below:

**Table 3-1064. Function tzpcu\_tzbmpc\_block\_secure\_access\_mode\_config**

<b>Function name</b>	tzpcu_tzbmpc_block_secure_access_mode_config
----------------------	--

<b>Function prototype</b>	void tzpcu_tzbmpc_block_secure_access_mode_config(uint32_t tzbmpx, uint32_t block_pos_num, uint32_t access_mode);
<b>Function descriptions</b>	configure block secure access mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>tzbmp</b>	TZBMPC
<i>TZBMPCx</i>	TZBMPCx(x=0,1,2,3)
<b>Input parameter{in}</b>	
<b>block_pos_num</b>	block position number(for TZBMPC0 and TZBMPC1 block position number is 0-255; for TZBMPC2 block position number is 0-511; for TZBMPC3 block position number is 0-799)
<b>Input parameter{in}</b>	
<b>access_mode</b>	bolck secure access mode
<i>BLOCK_SECURE_ACCESS_MODE_SEC</i>	bolck secure access mode is secure
<i>BLOCK_SECURE_ACCESS_MODE_NSEC</i>	bolck secure access mode is non-secure
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TZBMPC0 block 0 secure access mode to secure */
```

```
tzpcu_tzbmpc_block_secure_access_mode_config(TZBMPC0, 0, BLOCK_SECURE_ACCESS_MODE_SEC);
```

### tzpcu\_tzbmpc\_union\_block\_lock

The description of tzpcu\_tzbmpc\_union\_block\_lock is shown as below:

**Table 3-1065. Function tzpcu\_tzbmpc\_union\_block\_lock**

<b>Function name</b>	tzpcu_tzbmpc_union_block_lock
<b>Function prototype</b>	void tzpcu_tzbmpc_union_block_lock(uint32_t tzbmpx, uint32_t union_block_position_num);
<b>Function descriptions</b>	lock configuration union block secure access mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>tzbmp</b>	TZBMPC
<i>TZBMPCx</i>	TZBMPCx(x=0,1,2,3)

Input parameter{in}	
<b>union_block_position_num</b>	union block position number (for TZBMPC0 and TZBMPC1 union block position number is 0-7; for TZBMPC2 union block position number is 0-15; for TZBMPC3 union block position number is 0-23)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock configure union block secure access mode of TZBMPC0 union block 0 */
```

```
tzpcu_tzbmpc_union_block_lock(TZBMPC0, 0);
```

## tzpcu\_tziac\_interrupt\_enable

The description of tzpcu\_tziac\_interrupt\_enable is shown as below:

**Table 3-1066. Function tzpcu\_tziac\_interrupt\_enable**

<b>Function name</b>	tzpcu_tziac_interrupt_enable
<b>Function prototype</b>	void tzpcu_tziac_interrupt_enable(uint32_t periph);
<b>Function descriptions</b>	enable illegal access interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>periph</b>	peripheral
<i>TZPCU_PERIPH_TIME</i> <i>R1</i>	TIMER1 peripheral
<i>TZPCU_PERIPH_TIME</i> <i>R2</i>	TIMER2 peripheral
<i>TZPCU_PERIPH_TIME</i> <i>R3</i>	TIMER3 peripheral
<i>TZPCU_PERIPH_TIME</i> <i>R4</i>	TIMER4 peripheral
<i>TZPCU_PERIPH_TIME</i> <i>R5</i>	TIMER5 peripheral
<i>TZPCU_PERIPH_WW</i> <i>DGT</i>	WWDGT peripheral
<i>TZPCU_PERIPH_FWD</i> <i>GT</i>	FWDGT peripheral
<i>TZPCU_PERIPH_SPI1</i>	SPI1 peripheral
<i>TZPCU_PERIPH_USA</i> <i>RT1</i>	USART1 peripheral

TZPCU_PERIPH_USA RT2	USART2 peripheral
TZPCU_PERIPH_I2C0	I2C0 peripheral
TZPCU_PERIPH_I2C1	I2C1 peripheral
TZPCU_PERIPH_USB FS	USBFS peripheral
TZPCU_PERIPH_TIME R0	TIMER0 peripheral
TZPCU_PERIPH_SPI0	SPI0 peripheral
TZPCU_PERIPH_USA RT0	USART0 peripheral
TZPCU_PERIPH_TIME R15	TIMER15 peripheral
TZPCU_PERIPH_TIME R16	TIMER16 peripheral
TZPCU_PERIPH_HPD F	HPDF peripheral(HPDF not support on GD32W515Tx series devices)
TZPCU_PERIPH_CRC	CRC peripheral
TZPCU_PERIPH_TSI	TSI peripheral
TZPCU_PERIPH_ICAC HE	ICACHE peripheral
TZPCU_PERIPH_ADC	ADC peripheral
TZPCU_PERIPH_CAU	CAU peripheral
TZPCU_PERIPH_HAU	HAU peripheral
TZPCU_PERIPH_TRN G	TRNG peripheral
TZPCU_PERIPH_PKC AU	PKCAU peripheral
TZPCU_PERIPH_SDIO	SDIO peripheral
TZPCU_PERIPH_RTC	RTC peripheral
TZPCU_PERIPH_PMU	PMU peripheral
TZPCU_PERIPH_SYS CFG	SYSCFG peripheral
TZPCU_PERIPH_DMA 0	DMA0 peripheral
TZPCU_PERIPH_DMA 1	DMA1 peripheral
TZPCU_PERIPH_RCU	RCU peripheral
TZPCU_PERIPH_FLAS H	FLASH peripheral
TZPCU_PERIPH_FMC	FLASH REG peripheral
TZPCU_PERIPH_EXTI	EXTI peripheral
TZPCU_PERIPH_TZS	TZSPC peripheral

PC	
TZPCU_PERIPH_TZIA C	TZIAC peripheral
TZPCU_PERIPH_SRA M0	SRAM0 peripheral
TZPCU_PERIPH_TZB MPC0_REG	ZBMPC0 register
TZPCU_PERIPH_SRA M1	SRAM1 peripheral
TZPCU_PERIPH_TZB MPC1_REG	TZBMPC1 register
TZPCU_PERIPH_SRA M2	SRAM2 peripheral
TZPCU_PERIPH_TZB MPC2_REG	TZBMPC2 register
TZPCU_PERIPH_SRA M3	SRAM3 peripheral
TZPCU_PERIPH_TZB MPC3_REG	TZBMPC3 register
TZPCU_PERIPH_EFU SE	EFUSE peripheral
TZPCU_PERIPH_SQPI _PSRAM	SQPI_PSRAM peripheral
TZPCU_PERIPH_QSPI _FLASH	QSPI_FLASH peripheral
TZPCU_PERIPH_SQPI _PSRAMREG	SQPI PSRAMREG peripheral
TZPCU_PERIPH_QSPI _FLASHREG	QSPI FLASHREG peripheral
TZPCU_PERIPH_WIFI _RF	WIFI RF peripheral
TZPCU_PERIPH_I2S1 _ADD	I2S1_ADD peripheral
TZPCU_PERIPH_DCI	DCI peripheral(DCI not support on GD32W515Tx series devices)
TZPCU_PERIPH_WIFI	WIFI peripheral
TZPCU_PERIPH_ALL	all peripherals
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER1 illegal access interrupt */
```

```
tzpcu_tziac_interrupt_enable(TZPCU_PERIPH_TIMER1);
```

## tzpcu\_tziac\_interrupt\_disable

The description of tzpcu\_tziac\_interrupt\_disable is shown as below:

**Table 3-1067. Function tzpcu\_tziac\_interrupt\_disable**

Function name	tzpcu_tziac_interrupt_disable
Function prototype	void tzpcu_tziac_interrupt_disable (uint32_t periph);
Function descriptions	disable illegal access interrupt
Precondition	-
The called functions	-
Input parameter{in}	
periph	peripheral
TZPCU_PERIPH_TIMER1	TIMER1 peripheral
TZPCU_PERIPH_TIMER2	TIMER2 peripheral
TZPCU_PERIPH_TIMER3	TIMER3 peripheral
TZPCU_PERIPH_TIMER4	TIMER4 peripheral
TZPCU_PERIPH_TIMER5	TIMER5 peripheral
TZPCU_PERIPH_WWDGT	WWDGT peripheral
TZPCU_PERIPH_FWDGT	FWDGT peripheral
TZPCU_PERIPH_SPI1	SPI1 peripheral
TZPCU_PERIPH_USART1	USART1 peripheral
TZPCU_PERIPH_USART2	USART2 peripheral
TZPCU_PERIPH_I2C0	I2C0 peripheral
TZPCU_PERIPH_I2C1	I2C1 peripheral
TZPCU_PERIPH_USBFS	USBFS peripheral
TZPCU_PERIPH_TIMER0	TIMER0 peripheral
TZPCU_PERIPH_SPI0	SPI0 peripheral
TZPCU_PERIPH_USART0	USART0 peripheral
TZPCU_PERIPH_TIMER15	TIMER15 peripheral



TZPCU_PERIPH_TIME R16	TIMER16 peripheral
TZPCU_PERIPH_HPDI F	HPDI peripheral(HPDI not support on GD32W515Tx series devices)
TZPCU_PERIPH_CRC	CRC peripheral
TZPCU_PERIPH_TSI	TSI peripheral
TZPCU_PERIPH_ICACHE HE	ICACHE peripheral
TZPCU_PERIPH_ADC	ADC peripheral
TZPCU_PERIPH_CAU	CAU peripheral
TZPCU_PERIPH_HAU	HAU peripheral
TZPCU_PERIPH_TRNG G	TRNG peripheral
TZPCU_PERIPH_PKCAU AU	PKCAU peripheral
TZPCU_PERIPH_SDIO	SDIO peripheral
TZPCU_PERIPH_RTC	RTC peripheral
TZPCU_PERIPH_PMU	PMU peripheral
TZPCU_PERIPH_SYSCFG CFG	SYSCFG peripheral
TZPCU_PERIPH_DMA0 0	DMA0 peripheral
TZPCU_PERIPH_DMA1 1	DMA1 peripheral
TZPCU_PERIPH_RCU	RCU peripheral
TZPCU_PERIPH_FLASH H	FLASH peripheral
TZPCU_PERIPH_FLASH_REG FMC	FLASH REG peripheral
TZPCU_PERIPH_EXTI	EXTI peripheral
TZPCU_PERIPH_TZSPC PC	TZSPC peripheral
TZPCU_PERIPH_TZIAC C	TZIAC peripheral
TZPCU_PERIPH_SRAM0 M0	SRAM0 peripheral
TZPCU_PERIPH_ZBMPC0 MPC0_REG	ZBMPC0 register
TZPCU_PERIPH_SRAM1 M1	SRAM1 peripheral
TZPCU_PERIPH_ZBMPC1 MPC1_REG	ZBMPC1 register
TZPCU_PERIPH_SRAM2 M2	SRAM2 peripheral

<i>TZPCU_PERIPH_TZB MPC2_REG</i>	TZBMPC2 register
<i>TZPCU_PERIPH_SRA M3</i>	SRAM3 peripheral
<i>TZPCU_PERIPH_TZB MPC3_REG</i>	TZBMPC3 register
<i>TZPCU_PERIPH_EFU SE</i>	EFUSE peripheral
<i>TZPCU_PERIPH_SQPI _PSRAM</i>	SQPI_PSRAM peripheral
<i>TZPCU_PERIPH_QSPI _FLASH</i>	QSPI_FLASH peripheral
<i>TZPCU_PERIPH_SQPI _PSRAMREG</i>	SQPI PSRAMREG peripheral
<i>TZPCU_PERIPH_QSPI _FLASHREG</i>	QSPI FLASHREG peripheral
<i>TZPCU_PERIPH_WIFI _RF</i>	WIFI RF peripheral
<i>TZPCU_PERIPH_I2S1 _ADD</i>	I2S1_ADD peripheral
<i>TZPCU_PERIPH_DCI</i>	DCI peripheral(DCI not support on GD32W515Tx series devices)
<i>TZPCU_PERIPH_WIFI</i>	WIFI peripheral
<i>TZPCU_PERIPH_ALL</i>	all peripherals
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER1 illegal access interrupt */
```

```
tzpcu_tziac_interrupt_disable(TZPCU_PERIPH_TIMER1);
```

### **tzpcu\_tziac\_flag\_get**

The description of tzpcu\_tziac\_flag\_get is shown as below:

**Table 3-1068. Function tzpcu\_tziac\_flag\_get**

<b>Function name</b>	tzpcu_tziac_flag_get
<b>Function prototype</b>	uint32_t tzpcu_tziac_flag_get (uint32_tPeriph_flag);
<b>Function descriptions</b>	get illegal access interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>periph_flag</b>	peripheral interrupt flag
<i>TZPCU_TZIAC_FLAG_TIMER1</i>	TIMER1 peripheral
<i>TZPCU_TZIAC_FLAG_TIMER2</i>	TIMER2 peripheral
<i>TZPCU_TZIAC_FLAG_TIMER3</i>	TIMER3 peripheral
<i>TZPCU_TZIAC_FLAG_TIMER4</i>	TIMER4 peripheral
<i>TZPCU_TZIAC_FLAG_TIMER5</i>	TIMER5 peripheral
<i>TZPCU_TZIAC_FLAG_WWDGT</i>	WWDGT peripheral
<i>TZPCU_TZIAC_FLAG_FWDGT</i>	FWDGT peripheral
<i>TZPCU_TZIAC_FLAG_SPI1</i>	SPI1 peripheral
<i>TZPCU_TZIAC_FLAG_USART1</i>	USART1 peripheral
<i>TZPCU_TZIAC_FLAG_USART2</i>	USART2 peripheral
<i>TZPCU_TZIAC_FLAG_I2C0</i>	I2C0 peripheral
<i>TZPCU_TZIAC_FLAG_I2C1</i>	I2C1 peripheral
<i>TZPCU_TZIAC_FLAG_USBFS</i>	USBFS peripheral
<i>TZPCU_TZIAC_FLAG_TIMER0</i>	TIMER0 peripheral
<i>TZPCU_TZIAC_FLAG_SPI0</i>	SPI0 peripheral
<i>TZPCU_TZIAC_FLAG_USART0</i>	USART0 peripheral
<i>TZPCU_TZIAC_FLAG_TIMER15</i>	TIMER15 peripheral
<i>TZPCU_TZIAC_FLAG_TIMER16</i>	TIMER16 peripheral
<i>TZPCU_TZIAC_FLAG_HPDF</i>	HPDF peripheral(HPDF not support on GD32W515Tx series devices)
<i>TZPCU_TZIAC_FLAG_CRC</i>	CRC peripheral
<i>TZPCU_TZIAC_FLAG_TSI</i>	TSI peripheral

<i>TZPCU_TZIAC_FLAG_ICACHE</i>	ICACHE peripheral
<i>TZPCU_TZIAC_FLAG_ADC</i>	ADC peripheral
<i>TZPCU_TZIAC_FLAG_CAU</i>	CAU peripheral
<i>TZPCU_TZIAC_FLAG_HAU</i>	HAU peripheral
<i>TZPCU_TZIAC_FLAG_TRNG</i>	TRNG peripheral
<i>TZPCU_TZIAC_FLAG_PKCAU</i>	PKCAU peripheral
<i>TZPCU_TZIAC_FLAG_SDIO</i>	SDIO peripheral
<i>TZPCU_TZIAC_FLAG_RTC</i>	RTC peripheral
<i>TZPCU_TZIAC_FLAG_PMU</i>	PMU peripheral
<i>TZPCU_TZIAC_FLAG_SYSCFG</i>	SYSCFG peripheral
<i>TZPCU_TZIAC_FLAG_DMA0</i>	DMA0 peripheral
<i>TZPCU_TZIAC_FLAG_DMA1</i>	DMA1 peripheral
<i>TZPCU_TZIAC_FLAG_RCU</i>	RCU peripheral
<i>TZPCU_TZIAC_FLAG_FLASH</i>	FLASH peripheral
<i>TZPCU_TZIAC_FLAG_FMC</i>	FLASH REG peripheral
<i>TZPCU_TZIAC_FLAG_EXTI</i>	EXTI peripheral
<i>TZPCU_TZIAC_FLAG_TZSPC</i>	TZSPC peripheral
<i>TZPCU_TZIAC_FLAG_TZIAC</i>	TZIAC peripheral
<i>TZPCU_TZIAC_FLAG_SRAM0</i>	SRAM0 peripheral
<i>TZPCU_TZIAC_FLAG_TZBMPC0_REG</i>	ZBMPC0 register
<i>TZPCU_TZIAC_FLAG_SRAM1</i>	SRAM1 peripheral
<i>TZPCU_TZIAC_FLAG_TZBMPC1</i>	TZBMPC1 register

<i>TZBMPC1_REG</i>	
<i>TZPCU_TZIAC_FLAG_</i> <i>SRAM2</i>	SRAM2 peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>TZBMPC2_REG</i>	TZBMPC2 register
<i>TZPCU_TZIAC_FLAG_</i> <i>SRAM3</i>	SRAM3 peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>TZBMPC3_REG</i>	TZBMPC3 register
<i>TZPCU_TZIAC_FLAG_</i> <i>EFUSE</i>	EFUSE peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>SQPI_PSRAM</i>	SQPI_PSRAM peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>QSPI_FLASH</i>	QSPI_FLASH peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>SQPI_PSRAMREG</i>	SQPI PSRAMREG peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>QSPI_FLASHREG</i>	QSPI FLASHREG peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>WIFI_RF</i>	WIFI RF peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>I2S1_ADD</i>	I2S1_ADD peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>DCI</i>	DCI peripheral(DCI not support on GD32W515Tx series devices)
<i>TZPCU_TZIAC_FLAG_</i> <i>WIFI</i>	WIFI peripheral
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	NO_ILLEGAL_ACCESS_PENDING or ILLEGAL_ACCESS_PENDING

Example:

```
/* get TIMER1 illegal access interrupt flag */
```

```
uint32_t tziac_flag = tzpcu_tziac_flag_get(TZPCU_TZIAC_FLAG_TIMER1);
```

## **tzpcu\_tziac\_flag\_clear**

The description of tzpcu\_tziac\_flag\_clear is shown as below:

**Table 3-1069. Function tzpcu\_tziac\_flag\_clear**

<b>Function name</b>	tzpcu_tziac_flag_clear
<b>Function prototype</b>	void tzpcu_tziac_flag_clear (uint32_t periph_flag);

<b>Function descriptions</b>	clear illegal access interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph_flag</b>	peripheral interrupt flag
<i>TZPCU_TZIAC_FLAG_TIMER1</i>	TIMER1 peripheral
<i>TZPCU_TZIAC_FLAG_TIMER2</i>	TIMER2 peripheral
<i>TZPCU_TZIAC_FLAG_TIMER3</i>	TIMER3 peripheral
<i>TZPCU_TZIAC_FLAG_TIMER4</i>	TIMER4 peripheral
<i>TZPCU_TZIAC_FLAG_TIMER5</i>	TIMER5 peripheral
<i>TZPCU_TZIAC_FLAG_WWDGT</i>	WWDGT peripheral
<i>TZPCU_TZIAC_FLAG_FWDGT</i>	FWDGT peripheral
<i>TZPCU_TZIAC_FLAG_SPI1</i>	SPI1 peripheral
<i>TZPCU_TZIAC_FLAG_USART1</i>	USART1 peripheral
<i>TZPCU_TZIAC_FLAG_USART2</i>	USART2 peripheral
<i>TZPCU_TZIAC_FLAG_I2C0</i>	I2C0 peripheral
<i>TZPCU_TZIAC_FLAG_I2C1</i>	I2C1 peripheral
<i>TZPCU_TZIAC_FLAG_USBFS</i>	USBFS peripheral
<i>TZPCU_TZIAC_FLAG_TIMER0</i>	TIMER0 peripheral
<i>TZPCU_TZIAC_FLAG_SPI0</i>	SPI0 peripheral
<i>TZPCU_TZIAC_FLAG_USART0</i>	USART0 peripheral
<i>TZPCU_TZIAC_FLAG_TIMER15</i>	TIMER15 peripheral
<i>TZPCU_TZIAC_FLAG_TIMER16</i>	TIMER16 peripheral
<i>TZPCU_TZIAC_FLAG_HPDF</i>	HPDF peripheral(HPDF not support on GD32W515Tx series devices)

<i>TZPCU_TZIAC_FLAG_</i> <i>CRC</i>	CRC peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>TSI</i>	TSI peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>ICACHE</i>	ICACHE peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>ADC</i>	ADC peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>CAU</i>	CAU peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>HAU</i>	HAU peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>TRNG</i>	TRNG peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>PKCAU</i>	PKCAU peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>SDIO</i>	SDIO peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>RTC</i>	RTC peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>PMU</i>	PMU peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>SYSCFG</i>	SYSCFG peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>DMA0</i>	DMA0 peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>DMA1</i>	DMA1 peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>RCU</i>	RCU peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>FLASH</i>	FLASH peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>FMC</i>	FLASH REG peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>EXTI</i>	EXTI peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>TZSPC</i>	TZSPC peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>TZIAC</i>	TZIAC peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>SRAM0</i>	SRAM0 peripheral
<i>TZPCU_TZIAC_FLAG_</i>	ZBMPC0 register

<i>TZBMPC0_REG</i>	
<i>TZPCU_TZIAC_FLAG_</i> <i>SRAM1</i>	SRAM1 peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>TZBMPC1_REG</i>	TZBMPC1 register
<i>TZPCU_TZIAC_FLAG_</i> <i>SRAM2</i>	SRAM2 peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>TZBMPC2_REG</i>	TZBMPC2 register
<i>TZPCU_TZIAC_FLAG_</i> <i>SRAM3</i>	SRAM3 peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>TZBMPC3_REG</i>	TZBMPC3 register
<i>TZPCU_TZIAC_FLAG_</i> <i>EFUSE</i>	EFUSE peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>SQPI_PSRAM</i>	SQPI_PSRAM peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>QSPI_FLASH</i>	QSPI_FLASH peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>SQPI_PSRAMREG</i>	SQPI PSRAMREG peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>QSPI_FLASHREG</i>	QSPI FLASHREG peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>WIFI_RF</i>	WIFI RF peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>I2S1_ADD</i>	I2S1_ADD peripheral
<i>TZPCU_TZIAC_FLAG_</i> <i>DCI</i>	DCI peripheral(DCI not support on GD32W515Tx series devices)
<i>TZPCU_TZIAC_FLAG_</i> <i>WIFI</i>	WIFI peripheral
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER1 illegal access interrupt flag */
```

```
tzpcu_tziac_flag_clear(TZPCU_TZIAC_FLAG_TIMER1);
```



## 3.32. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.32.1](#), the USART firmware functions are introduced in chapter [3.32.2](#).

### 3.32.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

**Table 3-1070. USART Registers**

Registers	Descriptions
USART_CTL0	Control register 0
USART_CTL1	Control register 1
USART_CTL2	Control register 2
USART_BAUD	Baud rate register
USART_GP	Guard time and prescaler register
USART_RT	Receiver timeout register
USART_CMD	Command register
USART_STAT	Status register
USART_INTC	Status clear register
USART_RDATA	Receive data register
USART_TDATA	Transmit data register
USART_CHC	Coherence control register
USART_RFCS	Receive FIFO control and status register

### 3.32.2. Descriptions of Peripheral functions

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

**Table 3-1071. USART firmware function**

Function name	Function description
usart_deinit	reset USART
usart_baudrate_set	configure USART baud rate value
usart_parity_config	configure USART parity function
usart_word_length_set	configure USART word length
usart_stop_bit_set	configure USART stop bit length
usart_enable	enable USART
usart_disable	disable USART
usart_transmit_config	configure USART transmitter
usart_receive_config	configure USART receiver
usart_data_first_config	data is transmitted/received with the LSB/MSB first
usart_invert_config	configure USART inverted

Function name	Function description
usart_overrun_enable	enable the USART overrun function
usart_overrun_disable	disable the USART overrun function
usart_oversample_config	configure the USART oversample mode
usart_sample_bit_config	configure sample bit method
usart_receiver_timeout_enable	enable receiver timeout
usart_receiver_timeout_disable	disable receiver timeout
usart_receiver_timeout_threshold_config	configure receiver timeout threshold
usart_data_transmit	USART transmit data function
usart_data_receive	USART receive data function
usart_address_config	configure address of the USART
usart_address_detection_mode_config	configure address detection mode
usart_mute_mode_enable	enable mute mode
usart_mute_mode_disable	disable mute mode
usart_mute_mode_wakeup_config	configure wakeup method in mute mode
usart_lin_mode_enable	enable LIN mode
usart_lin_mode_disable	disable LIN mode
usart_lin_break_detection_length_config	LIN break detection length
usart_halfduplex_enable	enable half-duplex mode
usart_halfduplex_disable	disable half-duplex mode
usart_clock_enable	enable clock
usart_clock_disable	disable clock
usart_synchronous_clock_config	configure USART synchronous mode parameters
usart_guard_time_config	configure guard time value in smartcard mode
usart_smartcard_mode_enable	enable smartcard mode
usart_smartcard_mode_disable	disable smartcard mode
usart_smartcard_mode_nack_enable	enable NACK in smartcard mode
usart_smartcard_mode_nack_disable	disable NACK in smartcard mode
usart_smartcard_mode_early_nack_enable	enable early NACK in smartcard mode
usart_smartcard_mode_early_nack_disable	disable early NACK in smartcard mode
usart_smartcard_autoretry_config	configure smartcard auto-retry number
usart_block_length_config	configure block length
usart_irda_mode_enable	enable IrDA mode
usart_irda_mode_disable	disable IrDA mode
usart_prescaler_config	configure the peripheral clock prescaler in USART IrDA low-power or SmartCard mode
usart_irda_lowpower_config	configure IrDA low-power

Function name	Function description
usart_hardware_flow_rts_config	configure hardware flow control RTS
usart_hardware_flow_cts_config	configure hardware flow control CTS
usart_hardware_flow_coherence_config	configure hardware flow control coherence mode
usart_rs485_driver_enable	enable RS485 driver
usart_rs485_driver_disable	disable RS485 driver
usart_driver_asserttime_config	configure driver enable assertion time
usart_driver_deasserttime_config	configure driver enable de-assertion time
usart_depolarity_config	configure driver enable polarity mode
usart_dma_receive_config	configure USART DMA reception
usart_dma_transmit_config	configure USART DMA transmission
usart_reception_error_dma_disable	disable DMA on reception error
usart_reception_error_dma_enable	enable DMA on reception error
usart_wakeup_enable	enable USART to wakeup the MCU from deep-sleep mode
usart_wakeup_disable	disable USART to wakeup the MCU from deep-sleep mode
usart_wakeup_mode_config	configure the USART wakeup mode from deep-sleep mode
usart_receive_fifo_enable	enable receive FIFO
usart_receive_fifo_disable	disable receive FIFO
usart_receive_fifo_counter_number	read receive FIFO counter number
usart_command_enable	enable USART command
usart_flag_get	get flag in STAT/RFCFS register
usart_flag_clear	clear USART status
usart_interrupt_enable	enable USART interrupt
usart_interrupt_disable	disable USART interrupt
usart_interrupt_flag_get	get USART interrupt and flag status
usart_interrupt_flag_clear	clear USART interrupt flag

## Enum usart\_flag\_enum

Table 3-1072. Enum usart\_flag\_enum

Member name	Function description
USART_FLAG_REA	receive enable acknowledge flag
USART_FLAG_TEA	transmit enable acknowledge flag
USART_FLAG_WU	wakeup from Deep-sleep mode flag
USART_FLAG_RWU	receiver wakeup from mute mode
USART_FLAG_SB	send break flag
USART_FLAG_AM	ADDR match flag
USART_FLAG_BSY	busy flag
USART_FLAG_EB	end of block flag
USART_FLAG_RT	receiver timeout flag
USART_FLAG_CTS	CTS level
USART_FLAG_CTSF	CTS change flag

Member name	Function description
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_TBE	transmit data buffer empty
USART_FLAG_TC	transmission complete
USART_FLAG_RBNE	read data buffer not empty
USART_FLAG_IDLE	IDLE line detected flag
USART_FLAG_ORERR	overrun error
USART_FLAG_NERR	noise error flag
USART_FLAG_FERR	frame error flag
USART_FLAG_PERR	parity error flag
USART_FLAG_EPERR	early parity error flag
USART_FLAG_RFFINT	receive FIFO full interrupt flag
USART_FLAG_RFF	receive FIFO full flag
USART_FLAG_RFE	receive FIFO empty flag

## Enum usart\_interrupt\_flag\_enum

**Table 3-1073. Enum usart\_interrupt\_flag\_enum**

Member name	Function description
USART_INT_FLAG_EB	end of block interrupt and flag
USART_INT_FLAG_RT	receiver timeout interrupt and flag
USART_INT_FLAG_AM	address match interrupt and flag
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_ORERR	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_WU	wakeup from deep-sleep mode interrupt and flag
USART_INT_FLAG_CTS	CTS interrupt and flag
USART_INT_FLAG_ERR_NERR	error interrupt and noise error flag
USART_INT_FLAG_ERR_ORERR	error interrupt and overrun error
USART_INT_FLAG_ERR_FERR	error interrupt and frame error flag
USART_INT_FLAG_RFF	receive FIFO full interrupt and flag

## Enum usart\_interrupt\_enum

**Table 3-1074. Enum usart\_interrupt\_enum**

Member name	Function description
USART_INT_EB	end of block interrupt

Member name	Function description
USART_INT_RT	receiver timeout interrupt
USART_INT_AM	address match interrupt
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_WU	wakeup from deep-sleep mode interrupt
USART_INT_CTS	CTS interrupt
USART_INT_ERR	error interrupt
USART_INT_RFF	receive FIFO full interrupt

## Enum usart\_invert\_enum

**Table 3-1075. Enum usart\_invert\_enum**

Member name	Function description
USART_DINV_ENABLE	data bit level inversion
USART_DINV_DISABLE	data bit level not inversion
USART_TXPIN_ENABLE	TX pin level inversion
USART_TXPIN_DISABLE	TX pin level not inversion
USART_RXPIN_ENABLE	RX pin level inversion
USART_RXPIN_DISABLE	RX pin level not inversion
USART_SWAP_ENABLE	swap TX/RX pins
USART_SWAP_DISABLE	not swap TX/RX pins

## usart\_deinit

The description of usart\_deinit is shown as below:

**Table 3-1076. Function usart\_deinit**

<b>Function name</b>	usart_deinit
<b>Function prototype</b>	void usart_deinit(uint32_t usart_periph);
<b>Function descriptions</b>	reset USART
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<b>USARTx</b>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset USART0 */

usart_deinit(USART0);
```

## usart\_baudrate\_set

The description of usart\_baudrate\_set is shown as below:

**Table 3-1077. Function usart\_baudrate\_set**

<b>Function name</b>	usart_baudrate_set
<b>Function prototype</b>	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
<b>Function descriptions</b>	configure USART baud rate value
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>baudval</b>	baud rate value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 baud rate value */

usart_baudrate_set(USART0, 115200);
```

## usart\_parity\_config

The description of usart\_parity\_config is shown as below:

**Table 3-1078. Function usart\_parity\_config**

<b>Function name</b>	usart_parity_config
<b>Function prototype</b>	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
<b>Function descriptions</b>	configure USART parity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>paritycfg</b>	configure USART parity

<i>USART_PM_NONE</i>	no parity
<i>USART_PM_ODD</i>	odd parity
<i>USART_PM_EVEN</i>	even parity
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART parity */
usart_parity_config(USART0, USART_PM_EVEN);
```

## usart\_word\_length\_set

The description of usart\_word\_length\_set is shown as below:

**Table 3-1079. Function usart\_word\_length\_set**

<b>Function name</b>	usart_word_length_set
<b>Function prototype</b>	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
<b>Function descriptions</b>	configure USART word length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>wlen</b>	USART word length
<i>USART_WL_8BIT</i>	8 bits
<i>USART_WL_9BIT</i>	9 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 word length */
usart_word_length_set(USART0, USART_WL_9BIT);
```

## usart\_stop\_bit\_set

The description of usart\_stop\_bit\_set is shown as below:

**Table 3-1080. Function usart\_stop\_bit\_set**

<b>Function name</b>	usart_stop_bit_set
----------------------	--------------------

<b>Function prototype</b>	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
<b>Function descriptions</b>	configure USART stop bit length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>stblen</b>	USART stop bit
<i>USART_STB_1BIT</i>	1 bit
<i>USART_STB_0_5BIT</i>	0.5 bit
<i>USART_STB_2BIT</i>	2 bits
<i>USART_STB_1_5BIT</i>	1.5 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 stop bit length */
usart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

## usart\_enable

The description of usart\_enable is shown as below:

**Table 3-1081. Function usart\_enable**

<b>Function name</b>	usart_enable
<b>Function prototype</b>	void usart_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable USART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 */
usart_enable(USART0);
```



## usart\_disable

The description of usart\_disable is shown as below:

**Table 3-1082. Function usart\_disable**

<b>Function name</b>	usart_disable
<b>Function prototype</b>	void usart_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable USART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 */
usart_disable(USART0);
```

## usart\_transmit\_config

The description of usart\_transmit\_config is shown as below:

**Table 3-1083. Function usart\_transmit\_config**

<b>Function name</b>	usart_transmit_config
<b>Function prototype</b>	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
<b>Function descriptions</b>	configure USART transmitter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>txconfig</b>	enable or disable USART transmitter
<i>USART_TRANSMIT_ENABLE</i>	enable USART transmission
<i>USART_TRANSMIT_DISABLE</i>	disable USART transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

Example:

```
/* configure USART0 transmitter */
```

```
usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

### usart\_receive\_config

The description of usart\_receive\_config is shown as below:

**Table 3-1084. Function usart\_receive\_config**

<b>Function name</b>	usart_receive_config
<b>Function prototype</b>	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
<b>Function descriptions</b>	configure USART receiver
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
USARTx	x=0,1,2
<b>Input parameter{in}</b>	
<b>rxconfig</b>	enable or disable USART receiver
USART_RECEIVE_ENABLE	enable USART reception
USART_RECEIVE_DISABLE	disable USART reception
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 receiver */
```

```
usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

### usart\_data\_first\_config

The description of usart\_data\_first\_config is shown as below:

**Table 3-1085. Function usart\_data\_first\_config**

<b>Function name</b>	usart_data_first_config
<b>Function prototype</b>	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
<b>Function descriptions</b>	data is transmitted/received with the LSB/MSB first
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
Input parameter{in}	
<b>msbf</b>	LSB/MSB
<i>USART_MSBF_LSB</i>	LSB first
<i>USART_MSBF_MSB</i>	MSB first
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 data is transmitted/received with the LSB first */
```

```
usart_data_first_config(USART0, USART_MSBF_LSB);
```

### usart\_invert\_config

The description of usart\_invert\_config is shown as below:

**Table 3-1086. Function usart\_invert\_config**

<b>Function name</b>	usart_invert_config
<b>Function prototype</b>	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
<b>Function descriptions</b>	USART inverted configure
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
Input parameter{in}	
<b>invertpara</b>	refer to <a href="#">Table 3-1075. Enum usart_invert_enum</a>
<i>USART_DINV_ENABLER</i>	data bit level inversion
<i>USART_DINV_DISABLER</i>	data bit level not inversion
<i>USART_TXPIN_ENABLE</i>	TX pin level inversion
<i>USART_TXPIN_DISABLE</i>	TX pin level not inversion
<i>USART_RXPIN_ENABLE</i>	RX pin level inversion
<i>USART_RXPIN_DISABLE</i>	RX pin level not inversion

<i>LE</i>	
<i>USART_SWAP_ENAB</i> <i>LE</i>	swap TX/RX pins
<i>USART_SWAP_DISAB</i> <i>LE</i>	not swap TX/RX pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 data bit level inversion */
usart_invert_config (USART0, USART_DINV_ENABLE);
```

## usart\_oversrun\_enable

The description of usart\_oversrun\_enable is shown as below:

**Table 3-1087. Function usart\_oversrun\_enable**

<b>Function name</b>	usart_oversrun_enable
<b>Function prototype</b>	void usart_oversrun_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable the USART overrun function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the USART0 overrun function */
usart_oversrun_enable(USART0);
```

## usart\_oversrun\_disable

The description of usart\_oversrun\_disable is shown as below:

**Table 3-1088. Function usart\_oversrun\_disable**

<b>Function name</b>	usart_oversrun_disable
<b>Function prototype</b>	void usart_oversrun_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable the USART overrun function

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the USART0 overrun function */
```

```
usart_overrun_disable(USART0);
```

## usart\_oversample\_config

The description of usart\_oversample\_config is shown as below:

**Table 3-1089. Function usart\_oversample\_config**

<b>Function name</b>	usart_oversample_config
<b>Function prototype</b>	void usart_oversample_config(uint32_t usart_periph, uint32_t oversamp);
<b>Function descriptions</b>	configure the USART oversample mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>oversamp</b>	oversample value
<i>USART_OVSMOD_8</i>	8 bits
<i>USART_OVSMOD_16</i>	16 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 oversample mode */
```

```
usart_oversample_config(USART0, USART_OVSMOD_8);
```

## usart\_sample\_bit\_config

The description of usart\_sample\_bit\_config is shown as below:

Table 3-1090. Function `usart_sample_bit_config`

Function name	<code>usart_sample_bit_config</code>
Function prototype	<code>void usart_sample_bit_config(uint32_t usart_periph, uint32_t osb);</code>
Function descriptions	configure the sample bit method
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx peripheral
<code>USARTx</code>	x=0,1,2
Input parameter{in}	
<code>osb</code>	sample bit
<code>USART_OSB_1bit</code>	1 bits
<code>USART_OSB_3bit</code>	3 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 sample bit */
```

```
usart_sample_bit_config(USART0, USART_OSB_1bit);
```

### `usart_receiver_timeout_enable`

The description of `usart_receiver_timeout_enable` is shown as below:

Table 3-1091. Function `usart_receiver_timeout_enable`

Function name	<code>usart_receiver_timeout_enable</code>
Function prototype	<code>void usart_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=0,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable receiver timeout of USART */
```

```
usart_receiver_timeout_enable(USART0);
```

## usart\_receiver\_timeout\_disable

The description of usart\_receiver\_timeout\_disable is shown as below:

**Table 3-1092. Function usart\_receiver\_timeout\_disable**

<b>Function name</b>	usart_receiver_timeout_disable
<b>Function prototype</b>	void usart_receiver_timeout_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable receiver timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable receiver timeout of USART */
```

```
usart_receiver_timeout_disable(USART0);
```

## usart\_receiver\_timeout\_threshold\_config

The description of usart\_receiver\_timeout\_threshold\_config is shown as below:

**Table 3-1093. Function usart\_receiver\_timeout\_threshold\_config**

<b>Function name</b>	usart_receiver_timeout_threshold_config
<b>Function prototype</b>	void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);
<b>Function descriptions</b>	configure receiver timeout threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Input parameter{in}</b>	
<b>rtimeout</b>	receiver timeout threshold (0x00000000 - 0x00FFFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the receiver timeout threshold of USART0 */
```

```
usart_receiver_timeout_threshold_config(USART0, 115200*3);
```

## usart\_data\_transmit

The description of usart\_data\_transmit is shown as below:

**Table 3-1094. Function usart\_data\_transmit**

<b>Function name</b>	usart_data_transmit
<b>Function prototype</b>	void usart_data_transmit(uint32_t usart_periph, uint32_t data);
<b>Function descriptions</b>	USART transmit data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>data</b>	data of transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 transmit data */
```

```
usart_data_transmit(USART0, 0xAA);
```

## usart\_data\_receive

The description of usart\_data\_receive is shown as below:

**Table 3-1095. Function usart\_data\_receive**

<b>Function name</b>	usart_data_receive
<b>Function prototype</b>	uint16_t usart_data_receive(uint32_t usart_periph);
<b>Function descriptions</b>	USART receive data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	data of received



Example:

```
/* USART0 receive data */

uint16_t temp;

temp = usart_data_receive(USART0);
```

### usart\_address\_config

The description of usart\_address\_config is shown as below:

**Table 3-1096. Function usart\_address\_config**

<b>Function name</b>	usart_address_config
<b>Function prototype</b>	void usart_address_config(uint32_t usart_periph, uint8_t addr);
<b>Function descriptions</b>	address of the USART terminal
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
USARTx	x=0,1,2
<b>Input parameter{in}</b>	
<b>addr</b>	address of USART terminal(0x00-0xFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure address of USART0 */

usart_address_config(USART0, 0x00);
```

### usart\_address\_detection\_mode\_config

The description of usart\_address\_detection\_mode\_config is shown as below:

**Table 3-1097. Function usart\_address\_detection\_mode\_config**

<b>Function name</b>	usart_address_detection_mode_config
<b>Function prototype</b>	void usart_address_detection_mode_config(uint32_t usart_periph, uint32_t addmod);
<b>Function descriptions</b>	configure address detection mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
USARTx	x=0,1,2

Input parameter{in}	
<b>addmod</b>	address detection mode
<i>USART_ADDM_4BIT</i>	4 bits
<i>USART_ADDM_FULLBIT</i>	full bits
<i>IT</i>	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure address detection mode */
usart_address_config(USART0, USART_ADDM_4BIT);
```

### usart\_mute\_mode\_enable

The description of usart\_mute\_mode\_enable is shown as below:

**Table 3-1098. Function usart\_mute\_mode\_enable**

<b>Function name</b>	usart_mute_mode_enable
<b>Function prototype</b>	void usart_mute_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 receiver in mute mode */
usart_mute_mode_enable(USART0);
```

### usart\_mute\_mode\_disable

The description of usart\_mute\_mode\_disable is shown as below:

**Table 3-1099. Function usart\_mute\_mode\_disable**

<b>Function name</b>	usart_mute_mode_disable
<b>Function prototype</b>	void usart_mute_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable mute mode

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 receiver in mute mode */
```

```
usart_mute_mode_disable(USART0);
```

## usart\_mute\_mode\_wakeup\_config

The description of usart\_mute\_mode\_wakeup\_config is shown as below:

**Table 3-1100. Function usart\_mute\_mode\_wakeup\_config**

<b>Function name</b>	usart_mute_mode_wakeup_config
<b>Function prototype</b>	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
<b>Function descriptions</b>	configure wakeup method in mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>wmethod</b>	two methods be used to enter or exit the mute mode
<i>USART_WM_IDLE</i>	idle line
<i>USART_WM_ADDR</i>	address mark
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 wakeup method in mute mode */
```

```
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

## usart\_lin\_mode\_enable

The description of usart\_lin\_mode\_enable is shown as below:

**Table 3-1101. Function usart\_lin\_mode\_enable**

<b>Function name</b>	usart_lin_mode_enable
<b>Function prototype</b>	void usart_lin_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable LIN mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 LIN mode enable */
usart_lin_mode_enable(USART0);
```

## usart\_lin\_mode\_disable

The description of usart\_lin\_mode\_disable is shown as below:

**Table 3-1102. Function usart\_lin\_mode\_disable**

<b>Function name</b>	usart_lin_mode_disable
<b>Function prototype</b>	void usart_lin_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable LIN mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 LIN mode disable */
usart_lin_mode_disable(USART0);
```

## usart\_lin\_break\_dection\_length\_config

The description of usart\_lin\_break\_dection\_length\_config is shown as below:

**Table 3-1103. Function usart\_lin\_break\_dection\_length\_config**

<b>Function name</b>	usart_lin_break_dection_length_config
<b>Function prototype</b>	void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lblen);
<b>Function descriptions</b>	configure LIN break frame length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Input parameter{in}</b>	
<b>lblen</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LIN break frame length */
```

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

## usart\_halfduplex\_enable

The description of usart\_halfduplex\_enable is shown as below:

**Table 3-1104. Function usart\_halfduplex\_enable**

<b>Function name</b>	usart_halfduplex_enable
<b>Function prototype</b>	void usart_halfduplex_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable half duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 half duplex mode */
```

```
usart_halfduplex_enable(USART0);
```

### usart\_halfduplex\_disable

The description of usart\_halfduplex\_disable is shown as below:

**Table 3-1105. Function usart\_halfduplex\_disable**

<b>Function name</b>	usart_halfduplex_disable
<b>Function prototype</b>	void usart_halfduplex_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable half duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 half duplex mode*/
```

```
usart_halfduplex_disable(USART0);
```

### usart\_clock\_enable

The description of usart\_clock\_enable is shown as below:

**Table 3-1106. Function usart\_clock\_enable**

<b>Function name</b>	usart_clock_enable
<b>Function prototype</b>	void usart_clock_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable CK pin in synchronous mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 CK pin in synchronous mode */
```

```
usart_clock_enable(USART0);
```

### usart\_clock\_disable

The description of usart\_clock\_disable is shown as below:

**Table 3-1107. Function usart\_clock\_disable**

<b>Function name</b>	usart_clock_disable
<b>Function prototype</b>	void usart_clock_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable CK pin in synchronous mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 CK pin in synchronous mode */
```

```
usart_clock_disable(USART0);
```

### usart\_synchronous\_clock\_config

The description of usart\_synchronous\_clock\_config is shown as below:

**Table 3-1108. Function usart\_synchronous\_clock\_config**

<b>Function name</b>	usart_synchronous_clock_config
<b>Function prototype</b>	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
<b>Function descriptions</b>	configure USART synchronous mode parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Input parameter{in}</b>	
<b>clen</b>	CK length
<i>USART_CLEN_NONE</i>	clock pulse of the last data bit (MSB) is not output to the CK pin

<i>USART_CLEN_EN</i>	clock pulse of the last data bit (MSB) is output to the CK pin
<b>Input parameter{in}</b>	
<b>cph</b>	clock phase
<i>USART_CPH_1CK</i>	first clock transition is the first data capture edge
<i>USART_CPH_2CK</i>	second clock transition is the first data capture edge
<b>Input parameter{in}</b>	
<b>cpl</b>	clock polarity
<i>USART_CPL_LOW</i>	steady low value on CK pin
<i>USART_CPL_HIGH</i>	steady high value on CK pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 synchronous mode parameters */
```

```
usart_synchronous_clock_config(USART0,    USART_CLEN_EN,    USART_CPH_2CK,
USART_CPL_HIGH);
```

### usart\_guard\_time\_config

The description of usart\_guard\_time\_config is shown as below:

**Table 3-1109. Function usart\_guard\_time\_config**

<b>Function name</b>	usart_guard_time_config
<b>Function prototype</b>	void usart_guard_time_config(uint32_t usart_periph, uint32_t gaut);
<b>Function descriptions</b>	configure guard time value in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Input parameter{in}</b>	
<b>gaut</b>	guard time value (0x00 - 0xFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 guard time value in smartcard mode */
```

```
usart_guard_time_config(USART0, 0x55);
```



## usart\_smartcard\_mode\_enable

The description of usart\_smartcard\_mode\_enable is shown as below:

**Table 3-1110. Function usart\_smartcard\_mode\_enable**

<b>Function name</b>	usart_smartcard_mode_enable
<b>Function prototype</b>	void usart_smartcard_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 smartcard mode enable */
usart_smartcard_mode_enable(USART0);
```

## usart\_smartcard\_mode\_disable

The description of usart\_smartcard\_mode\_disable is shown as below:

**Table 3-1111. Function usart\_smartcard\_mode\_disable**

<b>Function name</b>	usart_smartcard_mode_disable
<b>Function prototype</b>	void usart_smartcard_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 smartcard mode disable */
usart_smartcard_mode_disable(USART0);
```

## usart\_smartcard\_mode\_nack\_enable

The description of usart\_smartcard\_mode\_nack\_enable is shown as below:

**Table 3-1112. Function usart\_smartcard\_mode\_nack\_enable**

<b>Function name</b>	usart_smartcard_mode_nack_enable
<b>Function prototype</b>	void usart_smartcard_mode_nack_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_enable(USART0);
```

## usart\_smartcard\_mode\_nack\_disable

The description of usart\_smartcard\_mode\_nack\_disable is shown as below:

**Table 3-1113. Function usart\_smartcard\_mode\_nack\_disable**

<b>Function name</b>	usart_smartcard_mode_nack_disable
<b>Function prototype</b>	void usart_smartcard_mode_nack_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_disable(USART0);
```

## usart\_smartcard\_mode\_early\_nack\_enable

The description of usart\_smartcard\_mode\_early\_nack\_enable is shown as below:

**Table 3-1114. Function usart\_smartcard\_mode\_early\_nack\_enable**

<b>Function name</b>	usart_smartcard_mode_early_nack_enable
<b>Function prototype</b>	void usart_smartcard_mode_early_nack_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable early NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 early NACK in smartcard mode */
usart_smartcard_mode_early_nack_enable(USART0);
```

## usart\_smartcard\_mode\_early\_nack\_disable

The description of usart\_smartcard\_mode\_early\_nack\_disable is shown as below:

**Table 3-1115. Function usart\_smartcard\_mode\_early\_nack\_disable**

<b>Function name</b>	usart_smartcard_mode_early_nack_disable
<b>Function prototype</b>	void usart_smartcard_mode_early_nack_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable early NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 early NACK in smartcard mode */
usart_smartcard_mode_early_nack_disable(USART0);
```

## usart\_smartcard\_autoretry\_config

The description of usart\_smartcard\_autoretry\_config is shown as below:

**Table 3-1116. Function usart\_smartcard\_autoretry\_config**

<b>Function name</b>	usart_smartcard_autoretry_config
<b>Function prototype</b>	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);
<b>Function descriptions</b>	configure smartcard auto-retry number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Input parameter{in}</b>	
<b>scrtnum</b>	smartcard auto-retry number (0x00 - 0x07)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure smartcard auto-retry number */
usart_smartcard_autoretry_config (USART0, 0x07);
```

## usart\_block\_length\_config

The description of usart\_block\_length\_config is shown as below:

**Table 3-1117. Function usart\_block\_length\_config**

<b>Function name</b>	usart_block_length_config
<b>Function prototype</b>	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
<b>Function descriptions</b>	configure block length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Input parameter{in}</b>	
<b>bl</b>	block length (0x00 - 0xFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure block length in Smartcard T=1 reception */
```

```
usart_block_length_config(USART0, 0x000000FF);
```

### usart\_irda\_mode\_enable

The description of usart\_irda\_mode\_enable is shown as below:

**Table 3-1118. Function usart\_irda\_mode\_enable**

<b>Function name</b>	usart_irda_mode_enable
<b>Function prototype</b>	void usart_irda_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable IrDA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 IrDA mode */
```

```
usart_irda_mode_enable(USART0);
```

### usart\_irda\_mode\_disable

The description of usart\_irda\_mode\_disable is shown as below:

**Table 3-1119. Function usart\_irda\_mode\_disable**

<b>Function name</b>	usart_irda_mode_disable
<b>Function prototype</b>	void usart_irda_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable IrDA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 IrDA mode */

usart_irda_mode_disable(USART0);
```

### usart\_prescaler\_config

The description of usart\_prescaler\_config is shown as below:

**Table 3-1120. Function usart\_prescaler\_config**

<b>Function name</b>	usart_prescaler_config
<b>Function prototype</b>	void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);
<b>Function descriptions</b>	configure the peripheral clock prescaler in USART IrDA low-power mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Input parameter{in}</b>	
<b>psc</b>	0x00000000 - 0x000000FF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the USART0 peripheral clock prescaler */

usart_prescaler_config(USART0, 0x00000000);
```

### usart\_irda\_lowpower\_config

The description of usart\_irda\_lowpower\_config is shown as below:

**Table 3-1121. Function usart\_irda\_lowpower\_config**

<b>Function name</b>	usart_irda_lowpower_config
<b>Function prototype</b>	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
<b>Function descriptions</b>	configure IrDA low-power
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,2
<b>Input parameter{in}</b>	
<b>irlp</b>	IrDA low-power or normal

<i>USART_IRLP_LOW</i>	low-power
<i>USART_IRLP_NORMAL</i>	normal
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

### usart\_hardware\_flow\_rts\_config

The description of usart\_hardware\_flow\_rts\_config is shown as below:

**Table 3-1122. Function usart\_hardware\_flow\_rts\_config**

<b>Function name</b>	usart_hardware_flow_rts_config
<b>Function prototype</b>	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
<b>Function descriptions</b>	configure hardware flow control RTS
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>rtsconfig</b>	enable or disable RTS
<i>USART_RTS_ENABLE</i>	enable RTS
<i>USART_RTS_DISABLE</i>	disable RTS
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 hardware flow control RTS */
```

```
usart_hardware_flow_cts_config(USART0, USART_RTS_ENABLE);
```

### usart\_hardware\_flow\_cts\_config

The description of usart\_hardware\_flow\_cts\_config is shown as below:

Table 3-1123. 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);
```

### usart\_hardware\_flow\_coherence\_config

The description of usart\_hardware\_flow\_coherence\_config is shown as below:

Table 3-1124. Function usart\_hardware\_flow\_coherence\_config

Function name	usart_hardware_flow_coherence_config
Function prototype	void usart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);
Function descriptions	configure hardware flow control coherence mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2
Input parameter{in}	
hcm	hardware flow control coherence mode
USART_HCM_NONE	nRTS signal equals to the rbne status register
USART_HCM_EN	nRTS signal is set when the last data bit has been sampled
Output parameter{out}	



-	-
Return value	
-	-

Example:

```
/* configure hardware flow control coherence mode */
```

```
usart_hardware_flow_coherence_config(USART0, USART_HCM_NONE);
```

### usart\_rs485\_driver\_enable

The description of usart\_rs485\_driver\_enable is shown as below:

**Table 3-1125. Function usart\_rs45\_driver\_enable**

<b>Function name</b>	usart_rs485_driver_enable
<b>Function prototype</b>	void usart_rs485_driver_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable RS485 driver
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 RS485 driver */
```

```
usart_rs485_driver_enable(USART0);
```

### usart\_rs485\_driver\_disable

The description of usart\_rs485\_driver\_disable is shown as below:

**Table 3-1126. Function usart\_rs45\_driver\_disable**

<b>Function name</b>	usart_rs485_driver_disable
<b>Function prototype</b>	void usart_rs485_driver_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable RS485 driver
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* disable USART0 RS485 driver */
usart_rs485_driver_disable(USART0);
```

### usart\_driver\_asserttime\_config

The description of usart\_driver\_asserttime\_config is shown as below:

**Table 3-1127. Function usart\_driver\_asserttime\_config**

<b>Function name</b>	usart_driver_asserttime_config
<b>Function prototype</b>	void usart_driver_asserttime_config(uint32_t usart_periph, uint32_t deatime);
<b>Function descriptions</b>	configure driver enable assertion time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<i>deatime</i>	driver enable assertion time(0x00-0x1F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set USART0 driver asserttime */
usart_driver_asserttime_config(USART0, 0x1F);
```

### usart\_driver\_deasserttime\_config

The description of usart\_driver\_deasserttime\_config is shown as below:

**Table 3-1128. Function usart\_driver\_deasserttime\_config**

<b>Function name</b>	usart_driver_deasserttime_config
<b>Function prototype</b>	void usart_driver_deasserttime_config(uint32_t usart_periph, uint32_t dedtime);
<b>Function descriptions</b>	configure driver enable de-assertion time
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
Input parameter{in}	
<i>deatime</i>	driver enable deasserttime (0x00-0x1F)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set USART0 driver deasserttime */
```

```
usart_driver_deasserttime_config(USART0, 0x1F);
```

## usart\_depolarity\_config

The description of usart\_depolarity\_config is shown as below:

**Table 3-1129. Function usart\_depolarity\_config**

<b>Function name</b>	usart_depolarity_config
<b>Function prototype</b>	void usart_depolarity_config(uint32_t usart_periph, uint32_t dep);
<b>Function descriptions</b>	configure driver enable polarity mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
Input parameter{in}	
<i>dep</i>	DE signal
<i>USART_DEP_HIGH</i>	DE signal is active high
<i>USART_DEP_LOW</i>	DE signal is active low
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure driver enable polarity mode */
```

```
usart_driver_depolarity_config(USART0, USART_DEP_HIGH);
```

## usart\_dma\_receive\_config

The description of usart\_dma\_receive\_config is shown as below:

**Table 3-1130. Function usart\_dma\_receive\_config**

<b>Function name</b>	usart_dma_receive_config
<b>Function prototype</b>	void usart_dma_receive_config(uint32_t usart_periph, uint8_t dmacmd);
<b>Function descriptions</b>	configure USART DMA reception
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
USARTx	x=0,1,2
<b>Input parameter{in}</b>	
<b>dmacmd</b>	USART DMA mode
USART_RECEIVE_DMA_ENABLE	enable USART DMA for reception
USART_RECEIVE_DMA_DISABLE	disable USART DMA for reception
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART DMA reception */
```

```
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

## usart\_dma\_transmit\_config

The description of usart\_dma\_transmit\_config is shown as below:

**Table 3-1131. Function usart\_dma\_transmit\_config**

<b>Function name</b>	usart_dma_transmit_config
<b>Function prototype</b>	void usart_dma_transmit_config(uint32_t usart_periph, uint8_t dmacmd);
<b>Function descriptions</b>	configure USART DMA transmission
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
USARTx	x=0,1,2
<b>Input parameter{in}</b>	
<b>dmacmd</b>	USART DMA mode
USART_TRANSMIT_DMA_ENABLE	enable USART DMA for transmission
USART_TRANSMIT_DMA_DISABLE	disable USART DMA for transmission

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART DMA transmission */
```

```
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

## usart\_reception\_error\_dma\_disable

The description of usart\_reception\_error\_dma\_disable is shown as below:

**Table 3-1132. Function usart\_reception\_error\_dma\_disable**

Function name	usart_reception_error_dma_disable
Function prototype	void usart_reception_error_dma_disable(uint32_t usart_periph);
Function descriptions	disable DMA on reception error
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA on reception error */
```

```
usart_reception_error_dma_disable (USART0);
```

## usart\_reception\_error\_dma\_enable

The description of usart\_reception\_error\_dma\_enable is shown as below:

**Table 3-1133. Function usart\_reception\_error\_dma\_enable**

Function name	usart_reception_error_dma_enable
Function prototype	void usart_reception_error_dma_enable(uint32_t usart_periph);
Function descriptions	enable DMA on reception error
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA on reception error */
```

```
usart_reception_error_dma_enable (USART0);
```

## usart\_wakeup\_enable

The description of usart\_wakeup\_enable is shown as below:

**Table 3-1134. Function usart\_wakeup\_enable**

Function name	usart_wakeup_enable
Function prototype	void usart_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	usart peripheral
USARTx	x=0,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 wake up enable */
```

```
usart_wakeup_enable(USART0);
```

## usart\_wakeup\_disable

The description of usart\_wakeup\_disable is shown as below:

**Table 3-1135. Function usart\_wakeup\_disable**

Function name	usart_wakeup_disable
Function prototype	void usart_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}	
usart_periph	usart peripheral
USARTx	x=0,2

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 wake up disable */
usart_wakeup_disable(USART0);
```

## usart\_wakeup\_mode\_config

The description of usart\_wakeup\_mode\_config is shown as below:

**Table 3-1136. Function usart\_wakeup\_mode\_config**

Function name	usart_wakeup_mode_config
Function prototype	void usart_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}	
usart_periph	usart peripheral
USARTx	x=0,2
Input parameter{in}	
wum	wakeup mode
USART_WUM_ADDR	WUF active on address match
USART_WUM_START B	WUF active on start bit
USART_WUM_RBNE	WUF active on RBNE
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART wake up mode */
usart_wakeup_mode_config(USART, USART_WUM_ADDR);
```

## usart\_receive\_fifo\_enable

The description of usart\_receive\_fifo\_enable is shown as below:

**Table 3-1137. Function usart\_receive\_fifo\_enable**

Function name	usart_receive_fifo_enable
Function prototype	void usart_receive_fifo_enable(uint32_t usart_periph);

<b>Function descriptions</b>	enable receive FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable receive FIFO */
usart_receive_fifo_enable(USART0);
```

### usart\_receive\_fifo\_disable

The description of usart\_receive\_fifo\_disable is shown as below:

**Table 3-1138. Function usart\_receive\_fifo\_disable**

<b>Function name</b>	usart_receive_fifo_disable
<b>Function prototype</b>	void usart_receive_fifo_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable receive FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable receive FIFO */
usart_receive_fifo_disable(USART0);
```

### usart\_receive\_fifo\_counter\_number

The description of usart\_receive\_fifo\_counter\_number is shown as below:

**Table 3-1139. Function usart\_receive\_fifo\_counter\_number**

<b>Function name</b>	usart_receive_fifo_counter_number
<b>Function prototype</b>	uint8_t usart_receive_fifo_counter_number(uint32_t usart_periph);



<b>Function descriptions</b>	read receive FIFO counter number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	receive FIFO counter number

Example:

```
/* read receive FIFO counter number */

uint8_t temp;

temp = usart_receive_fifo_counter_number(USART0);
```

## usart\_command\_enable

The description of usart\_command\_enable is shown as below:

**Table 3-1140. Function usart\_command\_enable**

<b>Function name</b>	usart_command_enable
<b>Function prototype</b>	void usart_command_enable(uint32_t usart_periph, uint32_t cmdtype);
<b>Function descriptions</b>	enable USART command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<i>cmdtype</i>	command type
<i>USART_CMD_SBKCM</i> <i>D</i>	send break command
<i>USART_CMD_MMCM</i> <i>D</i>	mute mode command
<i>USART_CMD_RXFCM</i> <i>D</i>	receive data flush command
<i>USART_CMD_TXFCM</i> <i>D</i>	transmit data flush request
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 command */
```

```
usart_command_enable(USART0, USART_CMD_SBKCMD);
```

## usart\_flag\_get

The description of usart\_flag\_get is shown as below:

**Table 3-1141. Function usart\_flag\_get**

<b>Function name</b>	usart_flag_get
<b>Function prototype</b>	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
<b>Function descriptions</b>	get flag in STAT/CHC/RFCR register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
USARTx	x=0,1,2
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to <a href="#">Table 3-1072. Enum usart_flag_enum</a>
USART_FLAG_PERR	parity error flag
USART_FLAG_FERR	frame error flag
USART_FLAG_NERR	noise error flag
USART_FLAG_ORER R	overrun error
USART_FLAG_IDLE	idle line detected flag
USART_FLAG_RBNE	read data buffer not empty
USART_FLAG_TC	transmission completed
USART_FLAG_TBE	transmit data register empty
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_CTSF	CTS change flag
USART_FLAG_CTS	CTS level
USART_FLAG_RT	receiver timeout flag
USART_FLAG_EB	end of block flag
USART_FLAG_BSY	busy flag
USART_FLAG_AM	address match flag
USART_FLAG_SB	send break flag
USART_FLAG_RWU	receiver wakeup from mute mode
USART_FLAG_WU	wakeup from deep-sleep mode flag
USART_FLAG_TEA	transmit enable acknowledge flag
USART_FLAG_REA	receive enable acknowledge flag
USART_FLAG_EPERR	early parity error flag
USART_FLAG_RFE	receive FIFO empty flag
USART_FLAG_RFF	receive FIFO full flag

USART_FLAG_RFFINT	receive FIFO full interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-1142. Function usart\_flag\_clear**

<b>Function name</b>	usart_flag_clear
<b>Function prototype</b>	void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);
<b>Function descriptions</b>	clear USART status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
usart_periph	USARTx peripheral
USARTx	x=0,1,2
<b>Input parameter{in}</b>	
flag	USART flags, refer to <a href="#">Table 3-1072. Enum usart_flag_enum</a>
USART_FLAG_PERR	parity error flag
USART_FLAG_FERR	frame error flag
USART_FLAG_NERR	noise detected flag
USART_FLAG_ORER R	overrun error flag
USART_FLAG_IDLE	idle line detected flag
USART_FLAG_TC	transmission complete flag
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_CTSF	CTS change flag
USART_FLAG_RT	receiver timeout flag
USART_FLAG_EB	end of block flag
USART_FLAG_AM	address match flag
USART_FLAG_WU	wakeup from deep-sleep mode flag
USART_FLAG_EPERR	early parity error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

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 3-1143. Function usart\_interrupt\_enable**

<b>Function name</b>	usart_interrupt_enable
<b>Function prototype</b>	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
<b>Function descriptions</b>	enable USART interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>interrupt</b>	USART interrupt, refer to <a href="#">Table 3-1074. Enum usart_interrupt_enum</a>
<i>USART_INT_IDLE</i>	idle interrupt
<i>USART_INT_RBNE</i>	read data buffer not empty interrupt and overrun error interrupt enable interrupt
<i>USART_INT_TC</i>	transmission complete interrupt
<i>USART_INT_TBE</i>	transmit data register empty interrupt
<i>USART_INT_PERR</i>	parity error interrupt
<i>USART_INT_AM</i>	address match interrupt
<i>USART_INT_RT</i>	receiver timeout interrupt
<i>USART_INT_EB</i>	end of block interrupt
<i>USART_INT_LBD</i>	LIN break detection interrupt
<i>USART_INT_ERR</i>	error interrupt enable in multibuffer communication
<i>USART_INT_CTS</i>	CTS interrupt
<i>USART_INT_WU</i>	wakeup from deep-sleep mode interrupt
<i>USART_INT_RFF</i>	receive FIFO full interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 TBE interrupt */
```

```
usart_interrupt_enable(USART0, USART_INT_TBE);
```

## usart\_interrupt\_disable

The description of usart\_interrupt\_disable is shown as below:

**Table 3-1144. Function usart\_interrupt\_disable**

<b>Function name</b>	usart_interrupt_disable
<b>Function prototype</b>	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
<b>Function descriptions</b>	disable USART interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>intterrupt</b>	USART interrupt flag, refer to <a href="#">Table 3-1074. Enum usart_interrupt_enum</a>
<i>USART_INT_IDLE</i>	idle interrupt
<i>USART_INT_RBNE</i>	read data buffer not empty interrupt and overrun error interrupt enable interrupt
<i>USART_INT_TC</i>	transmission complete interrupt
<i>USART_INT_TBE</i>	transmit data register empty interrupt
<i>USART_INT_PERR</i>	parity error interrupt
<i>USART_INT_AM</i>	address match interrupt
<i>USART_INT_RT</i>	receiver timeout interrupt
<i>USART_INT_EB</i>	end of block interrupt
<i>USART_INT_LBD</i>	LIN break detection interrupt
<i>USART_INT_ERR</i>	error interrupt enable in multibuffer communication
<i>USART_INT_CTS</i>	CTS interrupt
<i>USART_INT_WU</i>	wakeup from deep-sleep mode interrupt
<i>USART_INT_RFF</i>	receive FIFO full interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 TBE interrupt */
```

```
usart_interrupt_disable(USART0, USART_INT_TBE);
```

## usart\_interrupt\_flag\_get

The description of usart\_interrupt\_flag\_get is shown as below:

**Table 3-1145. Function usart\_interrupt\_flag\_get**

<b>Function name</b>	usart_interrupt_flag_get
<b>Function prototype</b>	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	get USART interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag, refer to <a href="#">Table 3-1073. Enum usart_interrupt_flag_enum</a>
USART_INT_FLAG_EB	end of block interrupt and flag
USART_INT_FLAG_RT	receiver timeout interrupt and flag
USART_INT_FLAG_A M	address match interrupt and flag
USART_INT_FLAG_PE RR	parity error interrupt and flag
USART_INT_FLAG_TB E	transmitter buffer empty interrupt and flag
USART_INT_FLAG_TC	transmission complete interrupt and flag
USART_INT_FLAG_RB NE	read data buffer not empty interrupt and flag
USART_INT_FLAG_RB NE_ORERR	read data buffer not empty interrupt and overrun error flag
USART_INT_FLAG_ID LE	IDLE line detected interrupt and flag
USART_INT_FLAG_LB D	LIN break detected interrupt and flag
USART_INT_FLAG_W U	wakeup from deep-sleep mode interrupt and flag
USART_INT_FLAG_CT S	CTS interrupt and flag
USART_INT_FLAG_ER R_NERR	error interrupt and noise error flag
USART_INT_FLAG_ER R_ORERR	error interrupt and overrun error

USART_INT_FLAG_ER R_FERR	error interrupt and frame error flag
USART_INT_FLAG_RF F	receive FIFO full 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 3-1146. 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 peripheral
USARTx	x=0,1,2
Input parameter{in}	
int_flag	USART interrupt flag, refer to <a href="#">Table 3-1073. Enum usart_interrupt_flag_enum</a>
USART_INT_FLAG_PE RR	parity error flag
USART_INT_FLAG_ER R_FERR	frame error flag
USART_INT_FLAG_ER R_NERR	noise detected flag
USART_INT_FLAG_RB NE_ORERR	read data buffer not empty interrupt and overrun error flag
USART_INT_FLAG_ER R_ORERR	error interrupt and overrun error
USART_INT_FLAG_ID LE	idle line detected flag

<code>USART_INT_FLAG_TC</code>	transmission complete flag
<code>USART_INT_FLAG_LBD</code>	LIN break detected flag
<code>USART_INT_FLAG_CTS</code>	CTS change flag
<code>USART_INT_FLAG_RT</code>	receiver timeout flag
<code>USART_INT_FLAG_EB</code>	end of block flag
<code>USART_INT_FLAG_AM</code>	address match flag
<code>USART_INT_FLAG_WU</code>	wakeup from deep-sleep mode flag
<code>USART_INT_FLAG_RFF</code>	receive FIFO full interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the USART0 interrupt enable bit status */
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

## 3.33. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.33.1](#), the FWDGT firmware functions are introduced in chapter [3.33.2](#).

### 3.33.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

**Table 3-1147. WWDGT Registers**

Registers	Descriptions
WWDGT_CTL	Control register
WWDGT_CFG	Configuration register
WWDGT_STAT	Status register

### 3.33.2. Descriptions of Peripheral functions

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



Table 3-1148. WWDGT firmware function

Function name	Function description
wwdgt_deinit	reset the WWDGT configuration
wwdgt_enable	start the WWDGT counter
wwdgt_counter_update	configure the WWDGT 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 3-1149. Function wwdgt\_deinit

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

Example:

```
/* reset the WWDGT configuration */
```

```
wwdgt_deinit ( );
```

### wwdgt\_enable

The description of wwdgt\_enable is shown as below:

Table 3-1150. Function wwdgt\_enable

Function name	wwdgt_enable
Function prototype	void wwdgt_enable(void);
Function descriptions	start the WWDGT 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 3-1151. 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(0x7F);
```

### wwdgt\_config

The description of wwdgt\_config is shown as below:

**Table 3-1152. 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}	

<b>window</b>	0x00 - 0x7F
<b>Input parameter{in}</b>	
<b>prescaler</b>	wwdgt prescaler value
WWDGT_CFG_PSC_D IV1	the time base of window watchdog counter = (PCLK1/4096)/1
WWDGT_CFG_PSC_D IV2	the time base of window watchdog counter = (PCLK1/4096)/2
WWDGT_CFG_PSC_D IV4	the time base of window watchdog counter = (PCLK1/4096)/4
WWDGT_CFG_PSC_D IV8	the time base of window watchdog counter = (PCLK1/4096)/8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(0x7F, 0x50, WWDGT_CFG_PSC_DIV8);
```

### wwdgt\_interrupt\_enable

The description of wwdgt\_interrupt\_enable is shown as below:

**Table 3-1153. Function wwdgt\_interrupt\_enable**

<b>Function name</b>	wwdgt_interrupt_enable
<b>Function prototype</b>	void wwdgt_interrupt_enable(void);
<b>Function descriptions</b>	enable early wakeup interrupt of WWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable early wakeup interrupt of WWDGT */
```

```
wwdgt_interrupt_enable ( );
```

## wwdgt\_flag\_get

The description of wwdgt\_flag\_get is shown as below:

**Table 3-1154. Function wwdgt\_flag\_get**

<b>Function name</b>	wwdgt_flag_get
<b>Function prototype</b>	FlagStatus wwdgt_flag_get(void);
<b>Function descriptions</b>	check early wakeup interrupt state of WWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get ( );
```

## wwdgt\_flag\_clear

The description of wwdgt\_flag\_clear is shown as below:

**Table 3-1155. Function wwdgt\_flag\_clear**

<b>Function name</b>	wwdgt_flag_clear
<b>Function prototype</b>	void wwdgt_flag_clear(void);
<b>Function descriptions</b>	clear early wakeup interrupt state of WWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear early wakeup interrupt state of WWDGT */
```

```
wwdgt_flag_clear( );
```

## 4. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Nov.23, 2021
1.1	<p>1. Change function  efuse_mcu_init_data_write /  efuse_aes_key_write /  efuse_rotpk_key_write / efuse_dp_write /  efuse_iak_write / efuse_user_data_write in  <u><b>3.8.2.</b></u></p> <p>2.Change function qspi_enable to  spi_quad_enable / qspi_disable to  spi_quad_disable / qspi_write_enable to  spi_quad_write_enable / qspi_read_enable  to spi_quad_read_enable /  qspi_io23_output_enable to  spi_quad_io23_output_enable /  qspi_io23_output_disable to  spi_quad_io23_output_disable in <u><b>3.24.2.</b></u></p> <p>3.Change function quad_spi_enable to  qspi_enable / quad_spi_disable to  qspi_disable in <u><b>3.20.2.</b></u></p> <p>4. Modify the function order in <u><b>3.21.2.</b></u></p>	Jul.5, 2022
2.0	Add GD32F5HC series and merge GD32W51x series with GD32F5HC series	Apr.15, 2026

## Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company 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.