

**GigaDevice Semiconductor Inc.**

**GD32VW55x  
RISC-V 32-bit MCU**

**Firmware Library  
User Guide**

Revision 1.1

(Jan. 2024)

# Table of Contents

<b>Table of Contents .....</b>	<b>2</b>
<b>List of Figures .....</b>	<b>5</b>
<b>List of Tables .....</b>	<b>6</b>
<b>1. Introduction .....</b>	<b>24</b>
<b>1.1. Rules of User Manual and Firmware Library .....</b>	<b>24</b>
1.1.1. Peripherals.....	24
1.1.2. Naming rules.....	25
<b>2. Firmware Library Overview.....</b>	<b>26</b>
<b>2.1. File Structure of Firmware Library .....</b>	<b>26</b>
2.1.1. Docs Folder.....	27
2.1.2. Examples Folder .....	28
2.1.3. Firmware Folder.....	28
2.1.4. Template Folder .....	29
2.1.5. Utilities Folder .....	32
<b>2.2. File descriptions of Firmware Library .....</b>	<b>32</b>
<b>3. Firmware Library of Standard Peripherals .....</b>	<b>34</b>
<b>3.1. Overview of Firmware Library of Standard Peripherals.....</b>	<b>34</b>
<b>3.2. ADC .....</b>	<b>34</b>
3.2.1. Descriptions of Peripheral registers.....	34
3.2.2. Descriptions of Peripheral functions .....	35
<b>3.3. CAU .....</b>	<b>60</b>
3.3.1. Descriptions of Peripheral registers.....	61
3.3.2. Descriptions of Peripheral functions .....	61
<b>3.4. CRC .....</b>	<b>88</b>
3.4.1. Descriptions of Peripheral registers.....	89
3.4.2. Descriptions of Peripheral functions .....	89
<b>3.5. DBG .....</b>	<b>93</b>
3.5.1. Descriptions of Peripheral registers.....	93
3.5.2. Descriptions of Peripheral functions .....	93
<b>3.6. DMA.....</b>	<b>97</b>
3.6.1. Descriptions of Peripheral registers.....	97
3.6.2. Descriptions of Peripheral functions .....	98
<b>3.7. ECLIC .....</b>	<b>123</b>
3.7.1. Descriptions of Peripheral functions .....	123

<b>3.8. EFUSE .....</b>	<b>129</b>
3.8.1. Descriptions of Peripheral registers .....	129
3.8.2. Descriptions of Peripheral functions .....	130
<b>3.9. EXTI .....</b>	<b>145</b>
3.9.1. Descriptions of Peripheral registers .....	145
3.9.2. Descriptions of Peripheral functions .....	146
<b>3.10. FMC .....</b>	<b>153</b>
3.10.1. Descriptions of Peripheral registers .....	153
3.10.2. Descriptions of Peripheral functions .....	154
<b>3.11. FWDGT .....</b>	<b>172</b>
3.11.1. Descriptions of Peripheral registers .....	172
3.11.2. Descriptions of Peripheral functions .....	173
<b>3.12. GPIO .....</b>	<b>177</b>
3.12.1. Descriptions of Peripheral registers .....	178
3.12.2. Descriptions of Peripheral functions .....	178
<b>3.13. HAU .....</b>	<b>188</b>
3.13.1. Descriptions of Peripheral registers .....	189
3.13.2. Descriptions of Peripheral functions .....	189
<b>3.14. I2C .....</b>	<b>208</b>
3.14.1. Descriptions of Peripheral registers .....	208
3.14.2. Descriptions of Peripheral functions .....	208
<b>3.15. PKCAU .....</b>	<b>247</b>
3.15.1. Descriptions of Peripheral registers .....	248
3.15.2. Descriptions of Peripheral functions .....	248
<b>3.16. PMU .....</b>	<b>278</b>
3.16.1. Descriptions of Peripheral registers .....	278
3.16.2. Descriptions of Peripheral functions .....	278
<b>3.17. QSPI .....</b>	<b>296</b>
3.17.1. Descriptions of Peripheral registers .....	296
3.17.2. Descriptions of Peripheral functions .....	296
<b>3.18. RCU .....</b>	<b>311</b>
3.18.1. Descriptions of Peripheral registers .....	311
3.18.2. Descriptions of Peripheral functions .....	312
<b>3.19. RTC .....</b>	<b>344</b>
3.19.1. Descriptions of Peripheral registers .....	345
3.19.2. Descriptions of Peripheral functions .....	345
<b>3.20. SPI .....</b>	<b>374</b>
3.20.1. Descriptions of Peripheral registers .....	374
3.20.2. Descriptions of Peripheral functions .....	374

<b>3.21.</b>	<b>SYSCFG .....</b>	<b>391</b>
3.21.1.	Descriptions of Peripheral registers .....	391
3.21.2.	Descriptions of Peripheral functions .....	391
<b>3.22.</b>	<b>TIMER.....</b>	<b>396</b>
3.22.1.	Descriptions of Peripheral registers .....	396
3.22.2.	Descriptions of Peripheral functions .....	397
<b>3.23.</b>	<b>TRNG.....</b>	<b>453</b>
3.23.1.	Descriptions of Peripheral registers .....	454
3.23.2.	Descriptions of Peripheral functions .....	454
<b>3.24.</b>	<b>USART.....</b>	<b>459</b>
3.24.1.	Descriptions of Peripheral registers .....	459
3.24.2.	Descriptions of Peripheral functions .....	460
<b>3.25.</b>	<b>WWDGT.....</b>	<b>507</b>
3.25.1.	Descriptions of Peripheral registers .....	507
3.25.2.	Descriptions of Peripheral functions .....	507
<b>4.</b>	<b>Revision history .....</b>	<b>512</b>

## List of Figures

Figure 2-1. File structure of firmware library of GD32VW55x.....	27
Figure 2-2. Select peripheral example files .....	29
Figure 2-3. Copy the peripheral example files .....	30
Figure 2-4. Open the project file .....	30
Figure 2-5. Configure project files .....	31
Figure 2-6. Compile.....	32

## List of Tables

Table 1-1. Peripherals .....	24
Table 2-1. Function descriptions of Firmware Library .....	32
Table 3-1. Peripheral function format of Firmware Library .....	34
Table 3-2. ADC Registers .....	34
Table 3-3. ADC firmware function.....	35
Table 3-4. Function adc_deinit.....	36
Table 3-5. Function adc_clock_config.....	37
Table 3-6. Function adc_enable .....	37
Table 3-7. Function adc_disable .....	38
Table 3-8. Function adc_dma_mode_enable .....	38
Table 3-9. Function adc_dma_mode_disable.....	39
Table 3-10. Function adc_dma_request_after_last_enable .....	39
Table 3-11. Function adc_dma_request_after_last_disable .....	40
Table 3-12. Function adc_discontinuous_mode_config .....	40
Table 3-13. Function adc_special_function_config .....	41
Table 3-14. Function adc_enable .....	42
Table 3-15. Function adc_disable.....	42
Table 3-16. Function adc_data_alignment_config.....	43
Table 3-17. Function adc_channel_length_config.....	43
Table 3-18. Function adc_routine_channel_config .....	44
Table 3-19. Function adc_inserted_channel_config .....	45
Table 3-20. Function adc_inserted_channel_offset_config.....	46
Table 3-21. Function adc_external_trigger_config.....	47
Table 3-22. Function adc_external_trigger_source_config .....	48
Table 3-23. Function adc_software_trigger_enable .....	49
Table 3-24. Function adc_end_of_conversion_config.....	50
Table 3-25. adc_resolution_config .....	50
Table 3-26. Function adc_routine_data_read.....	51
Table 3-27. Function adc_inserted_data_read .....	51
Table 3-28. Function adc_watchdog_single_channel_enable.....	52
Table 3-29. Function adc_watchdog_group_channel_enable.....	53
Table 3-30. Function adc_watchdog_disable .....	53
Table 3-31. Function adc_watchdog_threshold_config .....	54
Table 3-32. Function adc_oversample_mode_config .....	54
Table 3-33. Function adc_oversample_mode_enable .....	56
Table 3-34. Function adc_oversample_mode_disable .....	56
Table 3-35. Function adc_flag_get .....	57
Table 3-36. Function adc_flag_clear .....	57
Table 3-37. Function adc_interrupt_enable .....	58
Table 3-38. Function adc_interrupt_disable .....	59



Table 3-39. Function <code>adc_interrupt_flag_get</code> .....	59
Table 3-40. Function <code>adc_interrupt_flag_clear</code> .....	60
Table 3-41. CAU Registers .....	61
Table 3-42. CAU firmware function .....	61
Table 3-43. Structure <code>cau_key_parameter_struct</code> .....	62
Table 3-44. Structure <code>cau_iv_parameter_struct</code> .....	63
Table 3-45. Structure <code>cau_context_parameter_struct</code> .....	63
Table 3-46. Structure <code>cau_parameter_struct</code> .....	63
Table 3-47. Function <code>cau_deinit</code> .....	64
Table 3-48. Function <code>cau_struct_para_init</code> .....	64
Table 3-49. Function <code>cau_key_struct_para_init</code> .....	65
Table 3-50. Function <code>cau_iv_struct_para_init</code> .....	65
Table 3-51. Function <code>cau_context_struct_para_init</code> .....	66
Table 3-52. Function <code>cau_enable</code> .....	66
Table 3-53. Function <code>cau_disable</code> .....	67
Table 3-54. Function <code>cau_dma_enable</code> .....	67
Table 3-55. Function <code>cau_dma_disable</code> .....	68
Table 3-56. Function <code>cau_init</code> .....	68
Table 3-57. Function <code>cau_aes_keysize_config</code> .....	70
Table 3-58. Function <code>cau_key_init</code> .....	70
Table 3-59. Function <code>cau_iv_init</code> .....	71
Table 3-60. Function <code>cau_phase_config</code> .....	71
Table 3-61. Function <code>cau_fifo_flush</code> .....	72
Table 3-62. Function <code>cau_enable_state_get</code> .....	72
Table 3-63. Function <code>cau_data_write</code> .....	73
Table 3-64. Function <code>cau_data_read</code> .....	73
Table 3-65. Function <code>cau_context_save</code> .....	74
Table 3-66. Function <code>cau_context_restore</code> .....	75
Table 3-67. Function <code>cau_aes_ecb</code> .....	75
Table 3-68. Function <code>cau_aes_cbc</code> .....	76
Table 3-69. Function <code>cau_aes_ctr</code> .....	77
Table 3-70. Function <code>cau_aes_cfb</code> .....	78
Table 3-71. Function <code>cau_aes_ofb</code> .....	79
Table 3-72. Function <code>cau_aes_gcm</code> .....	80
Table 3-73. Function <code>cau_aes_ccm</code> .....	81
Table 3-74. Function <code>cau_tdes_ecb</code> .....	83
Table 3-75. Function <code>cau_tdes_cbc</code> .....	83
Table 3-76. Function <code>cau_des_ecb</code> .....	84
Table 3-77. Function <code>cau_des_cbc</code> .....	85
Table 3-78. Function <code>cau_flag_get</code> .....	86
Table 3-79. Function <code>cau_interrupt_enable</code> .....	87
Table 3-80. Function <code>cau_interrupt_disable</code> .....	87
Table 3-81. Function <code>cau_interrupt_flag_get</code> .....	88
Table 3-82. CRC Registers .....	89

Table 3-83. CRC firmware function .....	89
Table 3-84. Function crc_deinit.....	89
Table 3-85. Function crc_data_register_reset.....	90
Table 3-86. Function crc_data_register_read.....	90
Table 3-87. Function crc_free_data_register_read.....	91
Table 3-88. Function crc_free_data_register_write.....	91
Table 3-89. Function crc_single_data_calculate.....	92
Table 3-90. Function crc_block_data_calculate.....	92
Table 3-91. DBG Registers.....	93
Table 3-92. DBG firmware function .....	93
Table 3-93. Enum dbg_periph_enum .....	94
Table 3-94. Function dbg_deinit .....	94
Table 3-95. Function dbg_id_get .....	94
Table 3-96. Function dbg_low_power_enable.....	95
Table 3-97. Function dbg_low_power_disable.....	96
Table 3-98. Function dbg_periph_enable.....	96
Table 3-99. Function dbg_periph_disable.....	97
Table 3-100. DMA Registers.....	97
Table 3-101. DMA firmware function .....	98
Table 3-102. Enum dma_channel_enum .....	99
Table 3-103. Enum dma_subperipheral_enum.....	99
Table 3-104. Structure dma_multi_data_parameter_struct.....	100
Table 3-105. Structure dma_single_data_parameter_struct.....	100
Table 3-106. Function dma_deinit .....	100
Table 3-107. Function dma_single_data_para_struct_init .....	101
Table 3-108. Function dma_multi_data_para_struct_init .....	102
Table 3-109. Function dma_single_data_mode_init.....	102
Table 3-110. Function dma_multi_data_mode_init.....	103
Table 3-111. Function dma_periph_address_config .....	104
Table 3-112. Function dma_memory_address_config .....	105
Table 3-113. Function dma_transfer_number_config .....	106
Table 3-114. Function dma_transfer_number_get.....	106
Table 3-115. Function dma_priority_config .....	107
Table 3-116. Function dma_memory_burst_beats_config .....	107
Table 3-117. Function dma_periph_burst_beats_config .....	108
Table 3-118. Function dma_memory_width_config.....	109
Table 3-119. Function dma_periph_width_config.....	110
Table 3-120. Function dma_memory_address_generation_config .....	110
Table 3-121. Function dma_peripheral_address_generation_config.....	111
Table 3-122. Function dma_circulation_enable .....	112
Table 3-123. Function dma_circulation_disable .....	112
Table 3-124. Function dma_channel_enable .....	113
Table 3-125. Function dma_channel_disable.....	113
Table 3-126. Function dma_transfer_direction_config.....	114

Table 3-127. Function dma_switch_buffer_mode_config .....	114
Table 3-128. Function dma_using_memory_get.....	115
Table 3-129. Function dma_channel_subperipheral_select .....	116
Table 3-130. Function dma_flow_controller_config.....	116
Table 3-131. Function dma_switch_buffer_mode_enable.....	117
Table 3-132. Function dma_switch_buffer_mode_enable.....	118
Table 3-133. Function dma_fifo_status_get.....	118
Table 3-134. Function dma_flag_get.....	119
Table 3-135. Function dma_flag_clear .....	119
Table 3-136. Function dma_interrupt_enable.....	120
Table 3-137. Function dma_interrupt_disable.....	121
Table 3-138. Function dma_interrupt_flag_get .....	121
Table 3-139. Function dma_interrupt_flag_clear .....	122
Table 3-140. ECLIC firmware function .....	123
Table 3-141. Enum IRQn_Type .....	123
Table 3-142. Function eclic_global_interrupt_enable .....	125
Table 3-143. Function eclic_global_interrupt_disable .....	126
Table 3-144. Function eclic_level_threshold_set .....	126
Table 3-145. Function eclic_priority_group_set .....	127
Table 3-146. Function eclic_irq_enable .....	128
Table 3-147. Function eclic_irq_disable.....	128
Table 3-148. Function eclic_system_reset.....	129
Table 3-149. EFUSE Registers .....	129
Table 3-150. EFUSE firmware function .....	130
Table 3-151. Enum efuse_flag_enum.....	131
Table 3-152. Enum efuse_clear_flag_enum .....	131
Table 3-153. Enum efuse_int_enum .....	131
Table 3-154. Enum efuse_int_flag_enum .....	131
Table 3-155. efuse_clear_int_flag_enum.....	131
Table 3-156. Enum efuse_reg_lock_enum .....	132
Table 3-157. Function efuse_read .....	132
Table 3-158. Function efuse_write.....	133
Table 3-159. Function efuse_boot_config.....	133
Table 3-160. Function efuse_control1_config .....	134
Table 3-161. Function efuse_fp_config .....	134
Table 3-162. Function efuse_user_control_config.....	135
Table 3-163. Function efuse_res_write.....	136
Table 3-164. Function efuse_aes_key_write.....	136
Table 3-165. Function efuse_rotpk_key_write.....	137
Table 3-166. Function efuse_user_data_write.....	137
Table 3-167. Function efuse_res_read.....	138
Table 3-168. Function efuse_aes_key_read.....	138
Table 3-169. Function efuse_rotpk_key_read .....	139
Table 3-170. Function efuse_puid_read .....	140

Table 3-171. Function efuse_huk_key_read .....	140
Table 3-172. Function efuse_user_data_read .....	141
Table 3-173. Function efuse_boot_address_get.....	141
Table 3-174. Function efuse_lock_config .....	142
Table 3-175. Function efuse_flag_get.....	142
Table 3-176. Function efuse_flag_clear.....	143
Table 3-177. Function efuse_interrupt_enable.....	143
Table 3-178. Function efuse_interrupt_disable.....	144
Table 3-179. Function efuse_interrupt_flag_get .....	144
Table 3-180. Function efuse_interrupt_flag_clear .....	145
Table 3-181. EXTI Registers.....	145
Table 3-182. EXTI firmware function .....	146
Table 3-183. Enum exti_line_enum .....	146
Table 3-184. Enum exti_mode_enum .....	147
Table 3-185. Enum exti_trig_type_enum .....	147
Table 3-186. Function exti_deinit.....	147
Table 3-187. Function exti_init .....	148
Table 3-188. Function exti_interrupt_enable .....	148
Table 3-189. Function exti_interrupt_disable .....	149
Table 3-190. Function exti_event_enable.....	149
Table 3-191. Function exti_event_disable.....	150
Table 3-192. Function exti_software_interrupt_enable .....	150
Table 3-193. Function exti_software_interrupt_disable .....	151
Table 3-194. Function exti_flag_get .....	151
Table 3-195. Function exti_flag_clear .....	151
Table 3-196. Function exti_interrupt_flag_get.....	152
Table 3-197. Function exti_interrupt_flag_clear .....	152
Table 3-198. FMC Registers .....	153
Table 3-199. FMC firmware function .....	154
Table 3-200. fmc_state_enum.....	155
Table 3-201. Function fmc_unlock .....	155
Table 3-202. Function fmc_lock .....	155
Table 3-203. Function fmc_page_erase.....	156
Table 3-204. Function fmc_mass_erase.....	156
Table 3-205. Function fmc_word_program .....	157
Table 3-206. Function fmc_continuous_program.....	158
Table 3-207. Function fmc_obr_function_enable .....	158
Table 3-208. Function fmc_obr_function_disable .....	159
Table 3-209. Function ob_unlock .....	160
Table 3-210. Function ob_lock .....	160
Table 3-211. Function ob_start.....	161
Table 3-212. Function ob_reload .....	161
Table 3-213. Function ob_security_protection_config .....	162
Table 3-214. Function ob_user_write.....	162

Table 3-215. Function ob_write_protection_config .....	163
Table 3-216. Function fmc_no_rtdec_config .....	164
Table 3-217. Function fmc_offset_region_config .....	164
Table 3-218. Function fmc_offset_value_config .....	165
Table 3-219. Function fmc_wifi_trim_cal_get .....	166
Table 3-220. Function fmc_wifi_trim_pa_get .....	166
Table 3-221. Function fmc_wifi_trim_get .....	167
Table 3-222. Function ob_write_protection_get .....	167
Table 3-223. Function ob_user_get .....	168
Table 3-224. Function ob_security_protection_flag_get .....	168
Table 3-225. Function fmc_flag_get .....	169
Table 3-226. Function fmc_flag_clear .....	170
Table 3-227. Function fmc_interrupt_enable .....	170
Table 3-228. Function fmc_interrupt_disable .....	171
Table 3-229. Function fmc_interrupt_flag_get .....	171
Table 3-230. Function fmc_interrupt_flag_clear .....	172
Table 3-231. FWDGT Registers .....	173
Table 3-232. FWDGT firmware function .....	173
Table 3-233. Function fwdgt_write_enable .....	173
Table 3-234. Function fwdgt_write_disable .....	174
Table 3-235. Function fwdgt_enable .....	174
Table 3-236. Function fwdgt_prescaler_value_config .....	175
Table 3-237. Function fwdgt_reload_value_config .....	175
Table 3-238. Function fwdgt_counter_reload .....	176
Table 3-239. Function fwdgt_config .....	176
Table 3-240. Function fwdgt_flag_get .....	177
Table 3-241. GPIO Registers .....	178
Table 3-242. GPIO firmware function .....	178
Table 3-243. Function gpio_deinit .....	179
Table 3-244. Function gpio_mode_set .....	179
Table 3-245. Function gpio_output_options_set .....	180
Table 3-246. Function gpio_bit_set .....	181
Table 3-247. Function gpio_bit_reset .....	182
Table 3-248. Function gpio_bit_write .....	182
Table 3-249. Function gpio_port_write .....	183
Table 3-250. Function gpio_input_bit_get .....	183
Table 3-251. Function gpio_input_port_get .....	184
Table 3-252. Function gpio_output_bit_get .....	185
Table 3-253. Function gpio_output_port_get .....	185
Table 3-254. Function gpio_af_set .....	186
Table 3-255. Function gpio_pin_lock .....	187
Table 3-256. Function gpio_bit_toggle .....	187
Table 3-257. Function gpio_port_toggle .....	188
Table 3-258. HAU Registers .....	189

Table 3-259. HAU firmware function .....	189
Table 3-260. Structure hau_init_parameter_struct .....	190
Table 3-261. Structure hau_digest_parameter_struct .....	190
Table 3-262. Structure hau_context_parameter_struct .....	190
Table 3-263. Function hau_deinit .....	191
Table 3-264. Function hau_init .....	191
Table 3-265. Function hau_init_struct_para_init .....	192
Table 3-266. Function hau_reset .....	193
Table 3-267. Function hau_last_word_validbits_num_config .....	193
Table 3-268. Function hau_data_write .....	194
Table 3-269. Function hau_infifo_words_num_get .....	194
Table 3-270. Function hau_digest_read .....	195
Table 3-271. Function hau_digest_calculation_enable .....	195
Table 3-272. Function hau_multiple_single_dma_config .....	196
Table 3-273. Function hau_dma_enable .....	196
Table 3-274. Function hau_dma_disable .....	197
Table 3-275. Function hau_context_struct_para_init .....	197
Table 3-276. Function hau_context_save .....	198
Table 3-277. Function hau_context_restore .....	198
Table 3-278. Function hau_hash_sha_1 .....	199
Table 3-279. Function hau_hmac_sha_1 .....	199
Table 3-280. Function hau_hash_sha_224 .....	200
Table 3-281. Function hau_hmac_sha_224 .....	201
Table 3-282. Function hau_hash_sha_256 .....	201
Table 3-283. Function hau_hmac_sha_256 .....	202
Table 3-284. Function hau_hash_md5 .....	203
Table 3-285. Function hau_hmac_md5 .....	203
Table 3-286. Function hau_flag_get .....	204
Table 3-287. Function hau_flag_clear .....	205
Table 3-288. Function hau_interrupt_enable .....	205
Table 3-289. Function hau_interrupt_disable .....	206
Table 3-290. Function hau_interrupt_flag_get .....	207
Table 3-291. Function hau_interrupt_flag_clear .....	207
Table 3-292. I2C Registers .....	208
Table 3-293. I2C firmware function .....	208
Table 3-294. i2c_interrupt_flag_enum .....	210
Table 3-295. Function i2c_deinit .....	211
Table 3-296. Function i2c_timing_config .....	211
Table 3-297. Function i2c_digital_noise_filter_config .....	212
Table 3-298. Function i2c_analog_noise_filter_enable .....	213
Table 3-299. Function i2c_analog_noise_filter_disable .....	213
Table 3-300. Function i2c_master_clock_config .....	214
Table 3-301. Function i2c_master_addressing .....	214
Table 3-302. Function i2c_address10_header_enable .....	215

Table 3-303. Function i2c_address10_header_disable.....	216
Table 3-304. Function i2c_address10_enable .....	216
Table 3-305. Function i2c_address10_disable .....	217
Table 3-306. Function i2c_automatic_end_enable .....	217
Table 3-307. Function i2c_automatic_end_disable .....	218
Table 3-308. Function i2c_slave_response_to_gcall_enable .....	218
Table 3-309. Function i2c_slave_response_to_gcall_disable .....	219
Table 3-310. Function i2c_stretch_scl_low_enable .....	219
Table 3-311. Function i2c_stretch_scl_low_disable.....	220
Table 3-312. Function i2c_address_config .....	220
Table 3-313. Function i2c_address_bit_compare_config .....	221
Table 3-314. Function i2c_address_disable .....	222
Table 3-315. Function i2c_second_address_config.....	222
Table 3-316. Function i2c_second_address_disable .....	223
Table 3-317. Function i2c_receved_address_get .....	224
Table 3-318. Function i2c_slave_byte_control_enable.....	224
Table 3-319. Function i2c_slave_byte_control_disable.....	225
Table 3-320. Function i2c_nack_enable .....	225
Table 3-321. Function i2c_nack_disable .....	226
Table 3-322. Function i2c_wakeup_from_deepsleep_enable .....	226
Table 3-323. Function i2c_wakeup_from_deepsleep_disable .....	227
Table 3-324. Function i2c_enable .....	227
Table 3-325. Function i2c_disable .....	228
Table 3-326. Function i2c_start_on_bus .....	228
Table 3-327. Function i2c_stop_on_bus .....	229
Table 3-328. Function i2c_data_transmit .....	229
Table 3-329. Function i2c_data_receive .....	230
Table 3-330. Function i2c_reload_enable.....	231
Table 3-331. Function i2c_reload_disable .....	231
Table 3-332. Function i2c_transfer_byte_number_config.....	232
Table 3-333. Function i2c_dma_enable .....	232
Table 3-334. Function i2c_dma_disable .....	233
Table 3-335. Function i2c_pec_transfer .....	233
Table 3-336. Function i2c_pec_enable .....	234
Table 3-337. Function i2c_pec_disable .....	234
Table 3-338. Function i2c_pec_value_get.....	235
Table 3-339. Function i2c_smbus_alert_enable.....	235
Table 3-340. Function i2c_smbus_alert_disable.....	236
Table 3-341. Function i2c_smbus_default_addr_enable .....	236
Table 3-342. Function i2c_smbus_default_addr_disable .....	237
Table 3-343. Function i2c_smbus_host_addr_enable .....	237
Table 3-344. Function i2c_smbus_host_addr_disable .....	238
Table 3-345. Function i2c_extented_clock_timeout_enable.....	238
Table 3-346. Function i2c_extented_clock_timeout_disable.....	239

Table 3-347. Function i2c_clock_timeout_enable .....	239
Table 3-348. Function i2c_clock_timeout_disable .....	240
Table 3-349. Function i2c_bus_timeout_b_config.....	241
Table 3-350. Function i2c_bus_timeout_a_config .....	241
Table 3-351. Function i2c_idle_clock_timeout_config .....	242
Table 3-352. Function i2c_flag_get .....	242
Table 3-353. Function i2c_flag_clear .....	243
Table 3-354. Function i2c_interrupt_enable .....	244
Table 3-355. Function i2c_interrupt_disable .....	245
Table 3-356. Function i2c_interrupt_flag_get.....	245
Table 3-357. Function i2c_interrupt_flag_clear.....	247
Table 3-358. PKCAU Registers.....	248
Table 3-359. PKCAU firmware function .....	248
Table 3-360. Structure pkcau_mont_parameter_struct .....	249
Table 3-361. Structure pkcau_mod_parameter_struct .....	249
Table 3-362. Structure pkcau_mod_exp_parameter_struct.....	249
Table 3-363. Structure pkcau_arithmetic_parameter_struct .....	250
Table 3-364. Structure pkcau_crt_parameter_struct.....	250
Table 3-365. Structure pkcau_ec_group_parameter_struct .....	250
Table 3-366. Structure pkcau_point_parameter_struct .....	251
Table 3-367. Structure pkcau_signature_parameter_struct .....	251
Table 3-368. Structure pkcau_hash_parameter_struct.....	251
Table 3-369. Structure pkcau_ecc_out_struct.....	251
Table 3-370. Function pkcau_deinit .....	252
Table 3-371. Function pkcau_mont_struct_para_init.....	252
Table 3-372. Function pkcau_mod_struct_para_init.....	253
Table 3-373. Function pkcau_mod_exp_struct_para_init .....	253
Table 3-374. Function pkcau_arithmetic_struct_para_init.....	254
Table 3-375. Function pkcau_crt_struct_para_init.....	254
Table 3-376. Function pkcau_ec_group_struct_para_init.....	255
Table 3-377. Function pkcau_point_struct_para_init.....	255
Table 3-378. Function pkcau_signature_struct_para_init.....	256
Table 3-379. Function pkcau_hash_struct_para_init .....	257
Table 3-380. Function pkcau_ecc_out_struct_para_init .....	257
Table 3-381. Function pkcau_enable .....	258
Table 3-382. Function pkcau_disable .....	258
Table 3-383. Function pkcau_start .....	259
Table 3-384. Function pkcau_mode_set.....	259
Table 3-385. Function pkcau_mont_param_operation .....	260
Table 3-386. Function pkcau_mod_operation .....	261
Table 3-387. Function pkcau_mod_exp_operation .....	262
Table 3-388. Function pkcau_mod_inver_operation .....	263
Table 3-389. Function pkcau_mod_reduc_operation.....	264
Table 3-390. Function pkcau_arithmetic_operation .....	265

Table 3-391. Function pkcau crt_exp_operation .....	266
Table 3-392. Function pkcau_point_check_operation .....	267
Table 3-393. Function pkcau_point_mul_operation .....	268
Table 3-394. Function pkcau_ecdsa_sign_operation .....	270
Table 3-395. Function pkcau_ecdsa_verification_operation .....	272
Table 3-396. Function pkcau_flag_get .....	274
Table 3-397. Function pkcau_flag_clear .....	275
Table 3-398. Function pkcau_interrupt_enable .....	275
Table 3-399. Function pkcau_interrupt_disable .....	276
Table 3-400. Function pkcau_interrupt_flag_get .....	276
Table 3-401. Function pkcau_interrupt_flag_clear .....	277
Table 3-402. PMU Registers .....	278
Table 3-403. PMU firmware function .....	278
Table 3-404. Function pmu_deinit .....	279
Table 3-405. Function pmu_lvd_select .....	280
Table 3-406. Function pmu_lvd_disable .....	280
Table 3-407. Function pmu_backup_write_enable .....	281
Table 3-408. Function pmu_backup_write_disable .....	281
Table 3-409. Function pmu_to_sleepmode .....	282
Table 3-410. Function pmu_to_deepsleepmode .....	282
Table 3-411. Function pmu_to_standbymode .....	283
Table 3-412. Function pmu_wakeup_pin_enable .....	284
Table 3-413. Function pmu_wakeup_pin_disable .....	284
Table 3-414. Function pmu_wifi_power_enable .....	285
Table 3-415. Function pmu_wifi_power_disable .....	285
Table 3-416. Function pmu_wifi_sram_control .....	286
Table 3-417. Function pmu_ble_control .....	286
Table 3-418. Function pmu_ble_wakeup_request_enable .....	287
Table 3-419. Function pmu_ble_wakeup_request_disable .....	287
Table 3-420. Function pmu_pll_force_enable .....	288
Table 3-421. Function pmu_pll_force_disable .....	289
Table 3-422. Function pmu_ble_rf_config .....	289
Table 3-423. Function pmu_rf_force_enable .....	290
Table 3-424. Function pmu_rf_force_disable .....	290
Table 3-425. Function pmu_rf_sequence_config .....	291
Table 3-426. Function pmu_flag_get .....	291
Table 3-427. Function pmu_flag_clear .....	293
Table 3-428. Function pmu_interrupt_enable .....	293
Table 3-429. Function pmu_interrupt_disable .....	294
Table 3-430. Function pmu_interrupt_flag_get .....	294
Table 3-431. Function pmu_interrupt_flag_clear .....	295
Table 3-432. QSPI registers .....	296
Table 3-433. QSPI firmware function .....	296
Table 3-434. Structure qspi_init_struct .....	297



Table 3-435. Structure qspi_command_struct .....	297
Table 3-436. Structure qspi_polling_struct.....	298
Table 3-437. Function qspi_deinit .....	298
Table 3-438. Function qspi_struct_para_init .....	298
Table 3-439. Function qspi_cmd_struct_para_init .....	299
Table 3-440. Function qspi_polling_struct_para_init.....	300
Table 3-441. Function qspi_init.....	300
Table 3-442. Function qspi_enable .....	301
Table 3-443. Function qspi_disable .....	301
Table 3-444. Function qspi_dma_enable.....	302
Table 3-445. Function qspi_dma_disable.....	302
Table 3-446. Function qspi_command_config .....	303
Table 3-447. Function qspi_polling_config.....	304
Table 3-448. Function qspi_memorymapped_config.....	305
Table 3-449. Function qspi_data_transmit.....	306
Table 3-450. Function qspi_data_receive.....	306
Table 3-451. Function qspi_transmission_abort .....	307
Table 3-452. Function qspi_flag_get.....	307
Table 3-453. Function qspi_flag_clear.....	308
Table 3-454. Function qspi_interrupt_enable.....	309
Table 3-455. Function qspi_interrupt_disable.....	309
Table 3-456. Function qspi_interrupt_flag_get .....	310
Table 3-457. Function qspi_interrupt_flag_clear .....	311
Table 3-458. RCU Registers .....	311
Table 3-459. RCU firmware function .....	312
Table 3-460. Enum rcu_periph_enum .....	313
Table 3-461. Enum rcu_periph_reset_enum.....	314
Table 3-462. Enum rcu_unit_enum.....	315
Table 3-463. Enum rcu_flag_enum.....	315
Table 3-464. Enum rcu_int_flag_enum .....	316
Table 3-465. Enum rcu_int_flag_clear_enum .....	316
Table 3-466. Enum rcu_int_enum.....	317
Table 3-467. Enum rcu_osci_type_enum .....	317
Table 3-468. Enum rcu_clock_freq_enum.....	317
Table 3-469. Function rcu_deinit .....	318
Table 3-470. Function rcu_irc16m_dfs_to_rf_enable.....	318
Table 3-471. Function rcu_irc16m_dfs_to_rf_disable.....	319
Table 3-472. Function rcu_periph_clock_enable .....	319
Table 3-473. Function rcu_periph_clock_disable.....	320
Table 3-474. Function rcu_periph_clock_sleep_enable .....	320
Table 3-475. Function rcu_fmc_clock_sleep_disable.....	321
Table 3-476. Function rcu_periph_reset_enable.....	321
Table 3-477. Function rcu_periph_reset_disable .....	322
Table 3-478. Function rcu_bkp_reset_enable .....	322

Table 3-479. Function rcu_bkp_reset_disable .....	323
Table 3-480. Function rcu_rfppl_cal_enable .....	323
Table 3-481. Function rcu_rfppl_cal_disable .....	324
Table 3-482. Function rcu_control_unit_powerup .....	324
Table 3-483. Function rcu_control_unit_powerdown .....	325
Table 3-484. Function rcu_system_clock_source_config .....	325
Table 3-485. Function rcu_system_clock_source_get .....	326
Table 3-486. Function rcu_ahb_clock_config .....	326
Table 3-487. Function rcu_apb1_clock_config .....	327
Table 3-488. Function rcu_apb2_clock_config .....	328
Table 3-489. Function rcu_ckout0_config .....	328
Table 3-490. Function rcu_ckout1_config .....	329
Table 3-491. Function rcu_pll_config .....	330
Table 3-492. Function rcu_plldigdiv_sys_config .....	330
Table 3-493. Function rcu_rtc_clock_config .....	331
Table 3-494. Function rcu_rtc_div_config .....	332
Table 3-495. Function rcu_trng_div_config .....	332
Table 3-496. Function rcu_i2c0_clock_config .....	333
Table 3-497. Function rcu_usart0_clock_config .....	333
Table 3-498. Function rcu_irc16m_div_config .....	334
Table 3-499. Function rcu_sdio_div_config .....	334
Table 3-500. Function rcu_flag_get .....	335
Table 3-501. Function rcu_all_reset_flag_clear .....	336
Table 3-502. Function rcu_interrupt_flag_get .....	336
Table 3-503. Function rcu_interrupt_flag_clear .....	337
Table 3-504. Function rcu_interrupt_enable .....	337
Table 3-505. Function rcu_interrupt_disable .....	338
Table 3-506. Function rcu_lxtal_drive_capability_config .....	338
Table 3-507. Function rcu_osci_stab_wait .....	339
Table 3-508. Function rcu_osci_on .....	339
Table 3-509. Function rcu_osci_off .....	340
Table 3-510. Function rcu_osci_bypass_mode_enable .....	340
Table 3-511. Function rcu_osci_bypass_mode_disable .....	341
Table 3-512. Function rcu_rf_hxtal_clock_monitor_enable .....	341
Table 3-513. Function rcu_rf_hxtal_clock_monitor_disable .....	342
Table 3-514. Function rcu_irc16m_adjust_value_set .....	342
Table 3-515. Function rcu_voltage_key_unlock .....	343
Table 3-516. Function rcu_deepsleep_voltage_set .....	343
Table 3-517. Function rcu_clock_freq_get .....	344
Table 3-518. RTC Registers .....	345
Table 3-519. RTC firmware function .....	345
Table 3-520. Structure rtc_parameter_struct .....	347
Table 3-521. Structure rtc_alarm_struct .....	347
Table 3-522. Structure rtc_timestamp_struct .....	347

Table 3-523. Structure rtc_tamper_struct .....	347
Table 3-524. Function rtc_deinit .....	348
Table 3-525. Function rtc_init.....	348
Table 3-526. Function rtc_init_mode_enter .....	349
Table 3-527. Function rtc_init_mode_exit .....	350
Table 3-528. Function rtc_register_sync_wait .....	350
Table 3-529. Function rtc_current_time_get.....	351
Table 3-530. Function rtc_subsecond_get.....	351
Table 3-531. Function rtc_alarm_config.....	352
Table 3-532. Function rtc_alarm_subsecond_config.....	353
Table 3-533. Function rtc_alarm_get.....	354
Table 3-534. Function rtc_alarm_subsecond_get .....	354
Table 3-535. Function rtc_alarm_enable .....	355
Table 3-536. Function rtc_alarm_disable .....	356
Table 3-537. Function rtc_timestamp_enable .....	356
Table 3-538. Function rtc_timestamp_disable .....	357
Table 3-539. Function rtc_timestamp_get.....	357
Table 3-540. Function rtc_timestamp_subsecond_get.....	358
Table 3-541. Function rtc_tamper_enable.....	358
Table 3-542. Function rtc_tamper_disable.....	359
Table 3-543. Function rtc_software_bkp_reset.....	359
Table 3-544. Function rtc_tamper_without_bkp_seset.....	360
Table 3-545. Function rtc_output_pin_select .....	360
Table 3-546. Function rtc_alter_output_config .....	361
Table 3-547. Function rtc_calibration_output_config .....	362
Table 3-548. Function rtc_hour_adjust.....	362
Table 3-549. Function rtc_second_adjust.....	363
Table 3-550. Function rtc_bypass_shadow_enable .....	363
Table 3-551. Function rtc_bypass_shadow_disable .....	364
Table 3-552. Function rtc_refclock_detection_enable .....	364
Table 3-553. Function rtc_refclock_detection_disable .....	365
Table 3-554. Function rtc_wakeup_enable .....	365
Table 3-555. Function rtc_wakeup_disable .....	366
Table 3-556. Function rtc_wakeup_clock_set .....	366
Table 3-557. Function rtc_wakeup_timer_set.....	367
Table 3-558. Function rtc_wakeup_timer_get .....	368
Table 3-559. Function rtc_smooth_calibration_config .....	368
Table 3-560. Function rtc_coarse_calibration_enable .....	369
Table 3-561. Function rtc_coarse_calibration_disable.....	370
Table 3-562. Function rtc_coarse_calibration_config .....	370
Table 3-563. Function rtc_flag_get.....	371
Table 3-564. Function rtc_flag_clear.....	372
Table 3-565. Function rtc_interrupt_enable .....	373
Table 3-566. Function rtc_interrupt_disable.....	373

Table 3-567. SPI registers .....	374
Table 3-568. SPI firmware function .....	374
Table 3-569. Structure spi_parameter_struct .....	375
Table 3-570. Function spi_deinit .....	376
Table 3-571. Function spi_struct_para_init .....	376
Table 3-572. Function spi_init .....	377
Table 3-573. Function spi_enable .....	377
Table 3-574. Function spi_disable .....	378
Table 3-575. Function spi_nss_output_enable .....	378
Table 3-576. Function spi_nss_output_disable .....	379
Table 3-577. Function spi_nss_internal_high .....	379
Table 3-578. Function spi_nss_internal_low .....	380
Table 3-579. Function spi_dma_enable .....	380
Table 3-580. Function spi_dma_disable .....	381
Table 3-581. Function spi_data_frame_format_config .....	381
Table 3-582. Function spi_data_transmit .....	382
Table 3-583. Function spi_data_receive .....	382
Table 3-584. Function spi_bidirectional_transfer_config .....	383
Table 3-585. Function spi_format_error_clear .....	383
Table 3-586. Function spi_crc_polynomial_set .....	384
Table 3-587. Function spi_crc_polynomial_get .....	384
Table 3-588. Function spi_crc_on .....	385
Table 3-589. Function spi_crc_off .....	385
Table 3-590. Function spi_crc_next .....	386
Table 3-591. Function spi_crc_get .....	386
Table 3-592. Function spi_crc_error_clear .....	387
Table 3-593. Function spi_ti_mode_enable .....	387
Table 3-594. Function spi_ti_mode_disable .....	388
Table 3-595. Function spi_interrupt_enable .....	388
Table 3-596. Function spi_interrupt_disable .....	389
Table 3-597. Function spi_interrupt_flag_get .....	390
Table 3-598. Function spi_flag_get .....	390
Table 3-599. SYSCFG Registers .....	391
Table 3-600. SYSCFG firmware function .....	392
Table 3-601. Enum syscfg_code_start_enum .....	392
Table 3-602. Function syscfg_deinit .....	392
Table 3-603. Function syscfg_exti_line_config .....	393
Table 3-604. Function syscfg_io_compensation_enable .....	393
Table 3-605. Function syscfg_io_compensation_disable .....	394
Table 3-606. Function syscfg_lock_config .....	394
Table 3-607. Function syscfg_io_compensation_ready_flag_get .....	395
Table 3-608. Function syscfg_sram_ownership_config .....	395
Table 3-609. Function syscfg_boot_mode_get .....	396
Table 3-610. TIMERx Registers .....	397

Table 3-611. TIMERx firmware function .....	397
Table 3-612. Structure timer_parameter_struct .....	400
Table 3-613. Structure timer_break_parameter_struct .....	400
Table 3-614. Structure timer_oc_parameter_struct .....	400
Table 3-615. Structure timer_ic_parameter_struct .....	401
Table 3-616. Function timer_deinit .....	401
Table 3-617. Function timer_struct_para_init .....	402
Table 3-618. Function timer_init .....	402
Table 3-619. Function timer_enable .....	403
Table 3-620. Function timer_disable .....	403
Table 3-621. Function timer_auto_reload_shadow_enable .....	404
Table 3-622. Function timer_auto_reload_shadow_disable .....	405
Table 3-623. Function timer_update_event_enable .....	405
Table 3-624. Function timer_update_event_disable .....	406
Table 3-625. Function timer_counter_alignment .....	406
Table 3-626. Function timer_counter_up_direction .....	407
Table 3-627. timer_counter_down_direction .....	408
Table 3-628. Function timer_prescaler_config .....	408
Table 3-629. Function timer_repetition_value_config .....	409
Table 3-630. Function timer_autoreload_value_config .....	409
Table 3-631. Function timer_counter_value_config .....	410
Table 3-632. Function timer_counter_read .....	411
Table 3-633. Function timer_prescaler_read .....	411
Table 3-634. Function timer_single_pulse_mode_config .....	412
Table 3-635. Function timer_update_source_config .....	412
Table 3-636. Function timer_dma_enable .....	413
Table 3-637. Function timer_dma_disable .....	414
Table 3-638. Function timer_channel_dma_request_source_select .....	415
Table 3-639. Function timer_dma_transfer_config .....	415
Table 3-640. Function timer_event_software_generate .....	417
Table 3-641. Function timer_break_struct_para_init .....	418
Table 3-642. Function timer_break_config .....	418
Table 3-643. Function timer_break_enable .....	419
Table 3-644. Function timer_break_disable .....	420
Table 3-645. Function timer_automatic_output_enable .....	420
Table 3-646. Function timer_automatic_output_disable .....	421
Table 3-647. Function timer_primary_output_config .....	421
Table 3-648. Function timer_channel_control_shadow_config .....	422
Table 3-649. Function timer_channel_control_shadow_update_config .....	423
Table 3-650. Function timer_channel_output_struct_para_init .....	423
Table 3-651. Function timer_channel_output_config .....	424
Table 3-652. Function timer_channel_output_mode_config .....	425
Table 3-653. Function timer_channel_output_pulse_value_config .....	426
Table 3-654. Function timer_channel_output_shadow_config .....	427

Table 3-655. Function timer_channel_output_fast_config.....	428
Table 3-656. Function timer_channel_output_clear_config .....	428
Table 3-657. Function timer_channel_output_polarity_config.....	429
Table 3-658. Function timer_channel_complementary_output_polarity_config .....	430
Table 3-659. Function timer_channel_output_state_config .....	431
Table 3-660. Function timer_channel_complementary_output_state_config .....	432
Table 3-661. Function timer_channel_input_struct_para_init.....	432
Table 3-662. Function timer_input_capture_config.....	433
Table 3-663. Function timer_channel_input_capture_prescaler_config .....	434
Table 3-664. Function timer_channel_capture_value_register_read .....	435
Table 3-665. Function timer_input_pwm_capture_config .....	436
Table 3-666. Function timer_hall_mode_config.....	436
Table 3-667. Function timer_input_trigger_source_select .....	437
Table 3-668. Function timer_master_output_trigger_source_select .....	438
Table 3-669. Function timer_slave_mode_select .....	439
Table 3-670. Function timer_master_slave_mode_config .....	440
Table 3-671. Function timer_external_trigger_config.....	441
Table 3-672. Function timer_quadrature_decoder_mode_config.....	442
Table 3-673. Function timer_internal_clock_config .....	443
Table 3-674. Function timer_internal_trigger_as_external_clock_config .....	443
Table 3-675. Function timer_external_trigger_as_external_clock_config .....	444
Table 3-676. Function timer_external_clock_mode0_config.....	445
Table 3-677. Function timer_external_clock_mode1_config.....	446
Table 3-678. Function timer_external_clock_mode1_disable .....	447
Table 3-679. Function timer_write_chxval_register_config.....	447
Table 3-680. Function timer_output_value_selection_config .....	448
Table 3-681. Function timer_flag_get .....	449
Table 3-682. Function timer_flag_clear .....	450
Table 3-683. Function timer_interrupt_enable .....	450
Table 3-684. Function timer_interrupt_disable .....	451
Table 3-685. Function timer_interrupt_flag_get.....	452
Table 3-686. Function timer_interrupt_flag_clear.....	453
Table 3-687. TRNG Registers .....	454
Table 3-688. TRNG firmware function .....	454
Table 3-689. Enum trng_flag_enum .....	454
Table 3-690. Enum trng_int_flag_enum.....	454
Table 3-691. Function trng_deinit.....	455
Table 3-692. Function trng_enable.....	455
Table 3-693. Function trng_disable.....	456
Table 3-694. Function trng_get_true_random_data .....	456
Table 3-695. Function trng_interrupt_enable .....	457
Table 3-696. Function trng_interrupt_disable .....	457
Table 3-697. Function trng_flag_get .....	458
Table 3-698. Function trng_interrupt_flag_get.....	458

Table 3-699. Function <code>trng_interrupt_flag_clear</code> .....	459
Table 3-700. USART Registers .....	459
Table 3-701. USART firmware function.....	460
Table 3-702. Enum <code>usart_flag_enum</code> .....	462
Table 3-703. Enum <code>usart_interrupt_flag_enum</code> .....	462
Table 3-704. Enum <code>usart_interrupt_enum</code> .....	463
Table 3-705. Enum <code>usart_invert_enum</code> .....	463
Table 3-706. Function <code>usart_deinit</code> .....	464
Table 3-707. Function <code>usart_baudrate_set</code> .....	464
Table 3-708. Function <code>usart_parity_config</code> .....	465
Table 3-709. Function <code>usart_word_length_set</code> .....	465
Table 3-710. Function <code>usart_stop_bit_set</code> .....	466
Table 3-711. Function <code>usart_enable</code> .....	467
Table 3-712. Function <code>usart_disable</code> .....	467
Table 3-713. Function <code>usart_transmit_config</code> .....	468
Table 3-714. Function <code>usart_receive_config</code> .....	468
Table 3-715. Function <code>usart_data_first_config</code> .....	469
Table 3-716. Function <code>usart_invert_config</code> .....	470
Table 3-717. Function <code>usart_overrun_enable</code> .....	471
Table 3-718. Function <code>usart_overrun_disable</code> .....	471
Table 3-719. Function <code>usart_oversample_config</code> .....	472
Table 3-720. Function <code>usart_sample_bit_config</code> .....	472
Table 3-721. Function <code>usart_receiver_timeout_enable</code> .....	473
Table 3-722. Function <code>usart_receiver_timeout_disable</code> .....	473
Table 3-723. Function <code>usart_receiver_timeout_threshold_config</code> .....	474
Table 3-724. Function <code>usart_data_transmit</code> .....	474
Table 3-725. Function <code>usart_data_receive</code> .....	475
Table 3-726. Function <code>usart_command_enable</code> .....	476
Table 3-727. Function <code>usart_address_config</code> .....	476
Table 3-728. Function <code>usart_address_detection_mode_config</code> .....	477
Table 3-729. Function <code>usart_mute_mode_enable</code> .....	477
Table 3-730. Function <code>usart_mute_mode_disable</code> .....	478
Table 3-731. Function <code>usart_mute_mode_wakeup_config</code> .....	478
Table 3-732. Function <code>usart_lin_mode_enable</code> .....	479
Table 3-733. Function <code>usart_lin_mode_disable</code> .....	480
Table 3-734. Function <code>usart_lin_break_detection_length_config</code> .....	480
Table 3-735. Function <code>usart_halfduplex_enable</code> .....	481
Table 3-736. Function <code>usart_halfduplex_disable</code> .....	481
Table 3-737. Function <code>usart_clock_enable</code> .....	482
Table 3-738. Function <code>usart_clock_disable</code> .....	482
Table 3-739. Function <code>usart_synchronous_clock_config</code> .....	483
Table 3-740. Function <code>usart_guard_time_config</code> .....	484
Table 3-741. Function <code>usart_smartcard_mode_enable</code> .....	484
Table 3-742. Function <code>usart_smartcard_mode_disable</code> .....	485

Table 3-743. Function usart_smartcard_mode_nack_enable.....	485
Table 3-744. Function usart_smartcard_mode_nack_disable.....	486
Table 3-745. Function usart_smartcard_mode_early_nack_enable.....	486
Table 3-746. Function usart_smartcard_mode_early_nack_disable.....	487
Table 3-747. Function usart_smartcard_autoretry_config.....	487
Table 3-748. Function usart_block_length_config.....	488
Table 3-749. Function usart_irda_mode_enable.....	488
Table 3-750. Function usart_irda_mode_disable.....	489
Table 3-751. Function usart_prescaler_config.....	489
Table 3-752. Function usart_irda_lowpower_config.....	490
Table 3-753. Function usart_hardware_flow_rts_config.....	490
Table 3-754. Function usart_hardware_flow_cts_config.....	491
Table 3-755. Function usart_hardware_flow_coherence_config.....	492
Table 3-756. Function usart_rs45_driver_enable.....	492
Table 3-757. Function usart_rs45_driver_disable.....	493
Table 3-758. Function usart_driver_asserttime_config.....	493
Table 3-759. Function usart_driver_deasserttime_config.....	494
Table 3-760. Function usart_depolarity_config.....	495
Table 3-761. Function usart_dma_receive_config.....	495
Table 3-762. Function usart_dma_transmit_config.....	496
Table 3-763. Function usart_reception_error_dma_disable.....	497
Table 3-764. Function usart_reception_error_dma_enable.....	497
Table 3-765. Function usart_wakeup_enable.....	498
Table 3-766. Function usart_wakeup_disable.....	498
Table 3-767. Function usart_wakeup_mode_config.....	499
Table 3-768. Function usart_receive_fifo_enable.....	499
Table 3-769. Function usart_receive_fifo_disable.....	500
Table 3-770. Function usart_receive_fifo_counter_number.....	500
Table 3-771. Function usart_flag_get.....	501
Table 3-772. Function usart_flag_clear.....	502
Table 3-773. Function usart_interrupt_enable.....	503
Table 3-774. Function usart_interrupt_disable.....	504
Table 3-775. Function usart_interrupt_flag_get.....	505
Table 3-776. Function usart_interrupt_flag_clear.....	506
Table 3-777. WWDGT Registers.....	507
Table 3-778. WWDGT firmware function.....	508
Table 3-779. Function wwdgt_deinit.....	508
Table 3-780. Function wwdgt_enable.....	508
Table 3-781. Function wwdgt_counter_update.....	509
Table 3-782. Function wwdgt_config.....	509
Table 3-783. Function wwdgt_interrupt_enable.....	510
Table 3-784. Function wwdgt_flag_get.....	511
Table 3-785. Function wwdgt_flag_clear.....	511
Table 4-1. Revision history.....	512

## 1. Introduction

This manual introduces firmware library of GD32VW55x devices which are 32-bit microcontrollers based on the RISC-V processor.

The firmware library is a firmware function package, including program, data structure and macro definitions, all the performance features of peripherals of GD32VW55x 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.

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
DBG	Debug module
DMA	Direct memory access controller
EFUSE	Electronic fuse
EXTI	Interrupt/event controller
FMC	Flash memory controller
FWDGT	Free watchdog timer

Peripherals	Descriptions
GPIO/AFIO	General-purpose and alternate-function I/Os
HAU	Hash acceleration unit
I2C	Inter-integrated circuit interface
PKCAU	Public key cryptographic acceleration unit
PMU	Power management unit
QSPI	Quad-SPI interface
RCU	Reset and clock unit
RTC	Real-time Clock
SPI	Serial peripheral interface
TRNG	True random number generator
TIMER	TIMER
USART	Universal synchronous/asynchronous receiver /transmitter
WWDT	Window watchdog timer

### 1.1.2. Naming rules

The firmware library naming rules are shown as below:

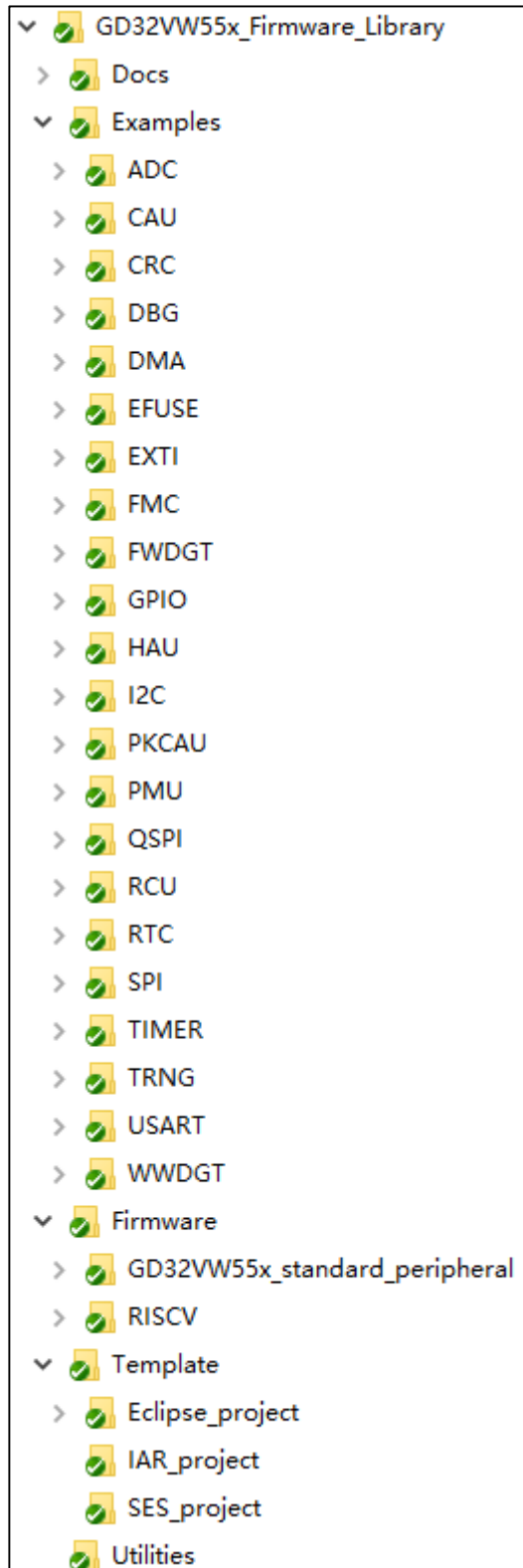
- The peripherals are shortened in XXX format, such as: ADC. More shorten information of peripherals refer to [Peripherals](#);
- The name of sourcefile and header file are started with “gd32vw55x\_”, such as: gd32vw55x\_adc.h;
- The constants used only in one file should be defined in the used file; the constants used in many files should be defined in corresponding header file. All the constants are written in uppercase of English letters;
- Registers are handled as constants. The naming of them are written in uppercase of English letters. In most cases, register names are shortened accord with the user manual;
- Variables are written in lowercase, when concluded by several words, underlines should be adapted among words;
- The naming of peripheral functions are started with the peripheral abbreviation added with an underline, when the function name is concluded by several words, underlines should be adapted among words, and all the peripheral functions are written in lowercase.

## **2. Firmware Library Overview**

### **2.1. File Structure of Firmware Library**

GD32VW55x\_Firmware\_Library, the file structure is shown as below:

Figure 2-1. File structure of firmware library of GD32VW55x



## 2.1.1. Docs Folder

The Docs folder contains the Firmware Library User Guide and development board schematic.

### 2.1.2. 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;
- `gd32vw55x_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);
- `gd32vw55x_it.c`: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- `gd32vw55x_it.h`: the header file include all the prototypes of the interrupt service routines;
- `systick.c`: the source file include the precise time delay functions by using systick;
- `systick.h`: the header file include the prototype of the precise time delay functions by using systick;
- `main.c`: example code.

**Note:** All the examples are not influenced by software IDEs.

### 2.1.3. Firmware Folder

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

- RISC-V subfolder:
  - Core subfolder includes the RISC-V kernel support files, users need not modify this folder;
  - `env_eclipse` subfolder includes the startup file based on the RISC-V kernel processor, exception service program and link script files of Eclipse IDE, users need not modify this folder;
  - `env_IAR` subfolder includes the startup file based on the RISC-V kernel processor and exception service program of IAR, users need not modify this folder;
  - `env_SES` subfolder includes the startup file based on the RISC-V kernel processor and exception service program of SES, users need not modify this folder;
  - `stubs` subfolder includes definition of pile functions such `_write/_read` function, users need not modify this folder.
- GD32VW55x\_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;
  - the global header file of GD32VW55x and system configuration file, users need not modify this folder.

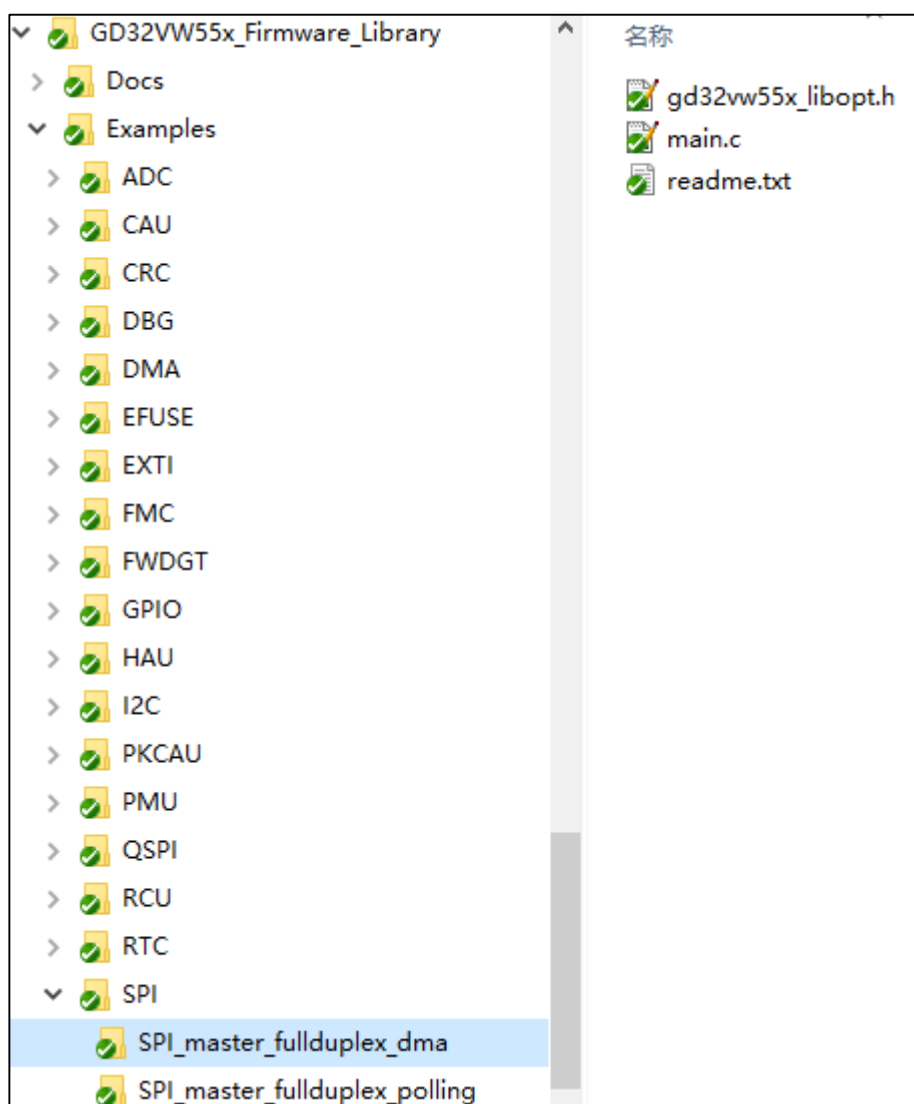
### 2.1.4. Template Folder

Template folder includes a simple demo of how to use LED\USART\KEY (IAR\_project run in EWRISCV-3101, Eclipse\_project run in GD32Eclipse, SES\_project run in SEGGER Embedded Studio for RISC-V 7.32a). User can use the project template to compile the formware examples, the steps are shown as below:

#### Select files

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

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

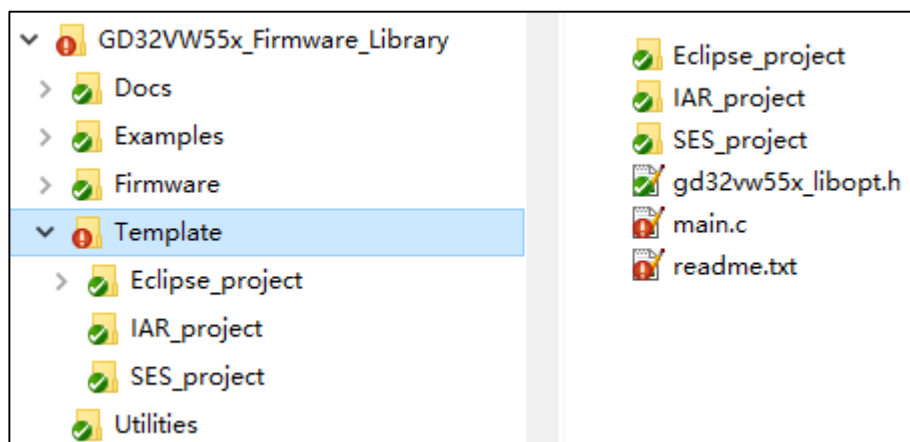


#### Copy files

Open “Template” folder, keep the folders of “IAR\_project”, “Eclipse\_project” and “SES\_project”, and delete the other files, then copy all the files in

“SPI\_master\_full duplex\_dma” folder to the “Template” subfolder, shown as below:

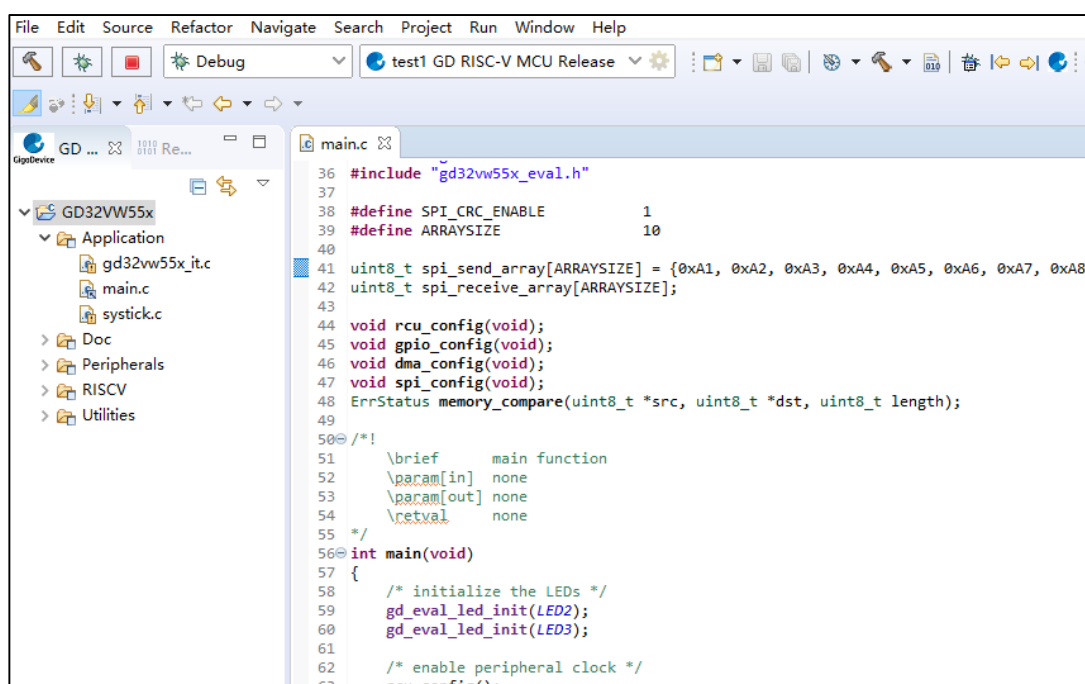
**Figure 2-3. Copy the peripheral example files**



## Open a project

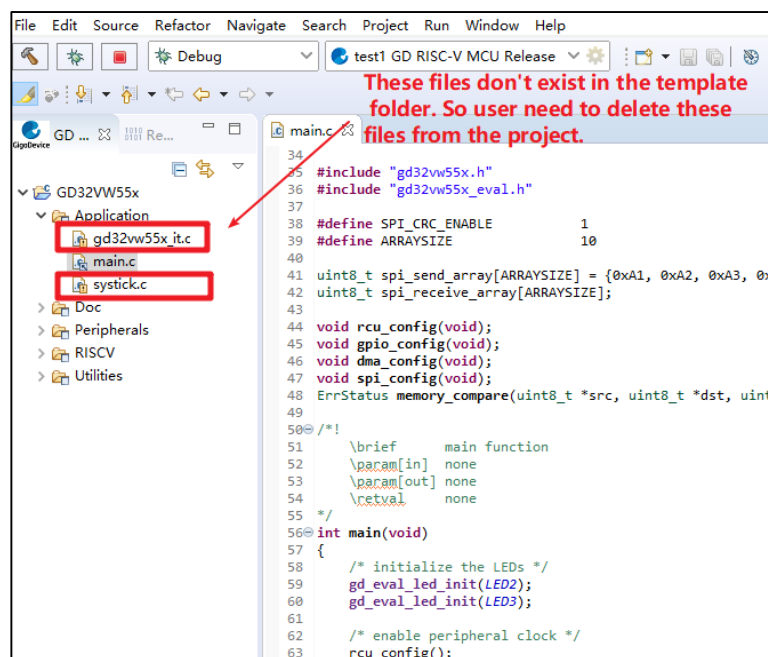
GD provides Eclipse, IAR and SES versions of the project. Depending on the software installed, user can open different projects. For example, open Eclipse and import project in Template folder, shown as below:

**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:

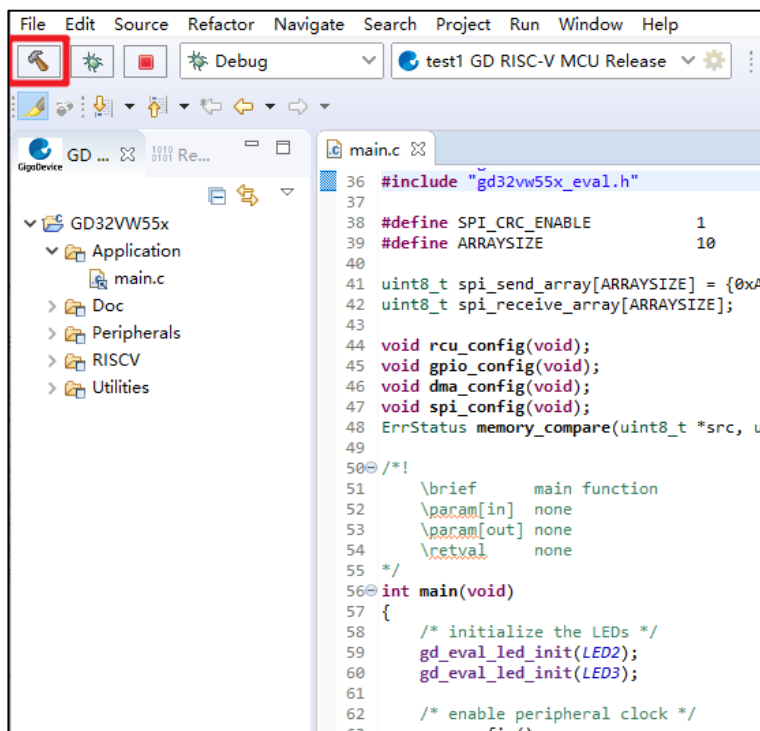
**Figure 2-5. Configure project files**



## 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. Compile operation is shown as below. Debug/download and other specific usage of IDE can refer to corresponding Eclipse software user guide.

Figure 2-6. Compile



## 2.1.5. Utilities Folder

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

- gd32vw55x\_eval.h is the related header file of the evaluation board about running the firmware examples;
- gd32vw55x\_eval.c is the related source file of the evaluation board about running the firmware examples.

## 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
gd32vw55x_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.
gd32vw55x_it.h	Header file, including all the prototypes of interrupt service routines.
gd32vw55x_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

Files	Descriptions
	by functions of judging interrupt flags of peripherals. The functions are included in the firmware library.
gd32vw55x_xxx.h	The header file of peripheral PPP, including functions about peripheral PPP, and the variables used for functions.
gd32vw55x_xxx.c	The C source file for driving peripheral PPP.
systick.h	The header file of systick.c, including prototypes of systick configuration function and delay function.
systick.c	The source file about systick configuration function and delay function.
readme.txt	Description document about how to configure and how to use the firmware example.

## 3. Firmware Library of Standard Peripherals

### 3.1. Overview of Firmware Library of Standard Peripherals

The description format of firmware functions are shown as below:

**Table 3-1. Peripheral function format of Firmware Library**

<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**

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

Registers	Descriptions
ADC_WDLT	Watchdog low threshold register
ADC_RSQ0	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_RDATA	Routine data 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 routine conversion
adc_dma_request_after_last_disable	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
adc_discontinuous_mode_config	configure ADC discontinuous mode
adc_special_function_config	enable or disable ADC special function
adc_tempsensor_vrefint_enable	enable temperature sensor and internal reference voltage channel
adc_tempsensor_vrefint_disable	disable temperature sensor and internal reference voltage channel
adc_data_alignment_config	configure ADC data alignment
adc_channel_length_config	configure the length of routine channel group or inserted channel group
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
adc_software_trigger_enable	enable ADC software trigger

Function name	Function description
adc_end_of_conversion_config	configure end of conversion mode
adc_resolution_config	configure ADC resolution
adc_routine_data_read	read ADC routine group data register
adc_inserted_data_read	read ADC routine group data register
adc_watchdog_single_channel_enable	configure ADC analog watchdog single channel
adc_watchdog_group_channel_enable	configure ADC analog watchdog group channel
adc_watchdog_disable	disable ADC analog watchdog
adc_watchdog_threshold_config	configure ADC analog watchdog threshold
adc_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(void);
<b>Function descriptions</b>	enable ADC DMA request

<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 DMA request */
```

```
adc_dma_mode_enable();
```

### **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(void);
<b>Function descriptions</b>	disable ADC DMA request
<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 DMA request */
```

```
adc_dma_mode_disable();
```

### **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(void);
<b>Function descriptions</b>	when DMA=1, the DMA engine issues a request at end of each routine conversion

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_dma_request_after_last_enable();
```

### **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(void);
<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>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_dma_request_after_last_enable();
```

### **adc\_discontinuous\_mode\_config**

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

**Table 3-12. 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_channel_group, 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_channel_group</b>	select the channel group
<i>ADC_ROUTINE_CHANNEL</i>	routine channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
<i>ADC_CHANNEL_DISCON_DISABLE</i>	disable discontinuous mode of routine and inserted channel
<b>Input parameter{in}</b>	
<b>length</b>	number of conversions in discontinuous mode, the number can be 1..9 for routine channel, the number has no effect for inserted channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC routine channel group 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 3-13. 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
<i>ADC_SCAN_MODE</i>	scan mode select
<i>ADC_INSERTED_CHANNEL_AUTO</i>	inserted channel group convert automatically
<i>ADC_CONTINUOUS_MODE</i>	continuous mode select
<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 ADC scan mode */
```

```
adc_special_function_config(ADC_SCAN_MODE, ENABLE);
```

### adc\_tempsensor\_vrefint\_enable

The description of adc\_tempsensor\_vrefint\_enable is shown as below:

**Table 3-14. Function adc\_enable**

<b>Function name</b>	adc_tempsensor_vrefint_enable
<b>Function prototype</b>	void adc_tempsensor_vrefint_enable (void);
<b>Function descriptions</b>	enable the temperature sensor and Vrefint channel
<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 temperature sensor and Vrefint channel */
```

```
adc_tempsensor_vrefint_enable ();
```

### adc\_tempsensor\_vrefint\_disable

The description of adc\_tempsensor\_vrefint\_disable is shown as below:

**Table 3-15. Function adc\_disable**

<b>Function name</b>	adc_tempsensor_vrefint_disable
<b>Function prototype</b>	void adc_tempsensor_vrefint_disable (void);
<b>Function descriptions</b>	disable the temperature sensor and Vrefint channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

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

```
adc_tempsensor_vrefint_disable ();
```

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

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>	adc_channel_length_config
<b>Function prototype</b>	void adc_channel_length_config(uint8_t adc_channel_group, uint32_t length);

<b>Function descriptions</b>	configure the length of routine channel group or inserted channel group
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_channel_group</b>	select the channel group
<i>ADC_ROUTINE_CHAN NEL</i>	routine channel group
<i>ADC_INSERTED_ CHANNEL</i>	inserted channel group
<b>Input parameter{in}</b>	
<b>length</b>	the length of the channel, routine channel 1-9, 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 group sequence rank, this parameter must be between 0 to 8
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the selected ADC channel
<i>ADC_CHANNEL_x (x=0..10)</i>	ADC Channelx (x=0..10)
<b>Input parameter{in}</b>	
<b>sample_time</b>	the sample time value
<i>ADC_SAMPLETIME_1 POINT5</i>	1.5 cycles
<i>ADC_SAMPLETIME_2 POINT5</i>	2.5 cycles

ADC_SAMPLETIME_1 4POINT5	14.5 cycles
ADC_SAMPLETIME_2 7POINT5	27.5 cycles
ADC_SAMPLETIME_5 5POINT5	55.5 cycles
ADC_SAMPLETIME_8 3POINT5	83.5 cycles
ADC_SAMPLETIME_1 11POINT5	111.5 cycles
ADC_SAMPLETIME_1 43POINT5	143.5 cycles
ADC_SAMPLETIME_4 79POINT5	479.5 cycles
Output parameter{out}	
-	-
Return value	
-	-

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>	-
Input parameter{in}	
<b>rank</b>	the inserted group sequencer rank, this parameter must be between 0 to 3
Input parameter{in}	
<b>adc_channel</b>	the selected ADC channel
ADC_CHANNEL_x (x=0..10)	ADC Channelx (x=0..10)
Input parameter{in}	
<b>sample_time</b>	the sample time value
ADC_SAMPLETIME_1	1.5 cycles

<i>POINT5</i>	
<i>ADC_SAMPLETIME_2</i> <i>POINT5</i>	2.5 cycles
<i>ADC_SAMPLETIME_1</i> <i>4POINT5</i>	14.5 cycles
<i>ADC_SAMPLETIME_2</i> <i>7POINT5</i>	27.5 cycles
<i>ADC_SAMPLETIME_5</i> <i>5POINT5</i>	55.5 cycles
<i>ADC_SAMPLETIME_8</i> <i>3POINT5</i>	83.5 cycles
<i>ADC_SAMPLETIME_1</i> <i>11POINT5</i>	111.5 cycles
<i>ADC_SAMPLETIME_1</i> <i>43POINT5</i>	143.5 cycles
<i>ADC_SAMPLETIME_4</i> <i>79POINT5</i>	479.5 cycles
<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>	<code>adc_inserted_channel_offset_config</code>
<b>Function prototype</b>	<code>void adc_inserted_channel_offset_config(uint8_t inserted_channel, uint16_t offset);</code>
<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

Output parameter{out}	
-	-
Return value	
-	-

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**

Function name	adc_external_trigger_config
Function prototype	void adc_external_trigger_config(uint8_t channel_group, uint32_t trigger_mode);
Function descriptions	configure ADC external trigger
Precondition	-
The called functions	-
Input parameter{in}	
<b>adc_channel_group</b>	select the channel group
ADC_ROUTINE_CHANNEL	routine channel group
ADC_INSERTED_CHANNEL	inserted channel group
Input parameter{in}	
<b>trigger_mode</b>	external trigger mode
EXTERNAL_TRIGGER_DISABLE	external trigger disable
EXTERNAL_TRIGGER_RISING	rising edge of external trigger
EXTERNAL_TRIGGER_FALLING	falling edge of external trigger
EXTERNAL_TRIGGER_RISING_FALLING	rising and falling edge of external trigger
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC inserted channel group 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>	adc_external_trigger_source_config
<b>Function prototype</b>	void adc_external_trigger_source_config(uint8_t channel_group, uint32_t external_trigger_source);
<b>Function descriptions</b>	configure ADC external trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_channel_group</b>	select the channel group
ADC_ROUTINE_CHANNEL	routine channel group
ADC_INSERTED_CHANNEL	inserted channel group
<b>Input parameter{in}</b>	
<b>external_trigger_source</b>	routine or inserted group trigger 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_EXTI_11	extiline 11 event select for routine channel
ADC_EXTTRIG_INSERTED_T0_CH3	TIMER0 CH3 event select for inserted channel

<i>INSERTED_T0_CH3</i>	
<i>ADC_EXTTRIG_</i> <i>INSERTED_T0_TRGO</i>	TIMER0 TRGO event select for inserted channel
<i>ADC_EXTTRIG_</i> <i>INSERTED_T1_CH0</i>	TIMER1 CH0 event select for inserted channel
<i>ADC_EXTTRIG_</i> <i>INSERTED_T1_TRGO</i>	TIMER1 TRGO event select for inserted channel
<i>ADC_EXTTRIG_</i> <i>INSERTED_T2_CH1</i>	TIMER2 CH1 event select for inserted channel
<i>ADC_EXTTRIG_</i> <i>INSERTED_T2_CH3</i>	TIMER3 CH3 event select for inserted channel
<i>ADC_EXTTRIG_</i> <i>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_channel_group);</code>
<b>Function descriptions</b>	enable ADC software trigger
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_channel_group</b>	select the channel group
<i>ADC_ROUTINE_</i> <i>CHANNEL</i>	routine channel group
<i>ADC_INSERTED_</i> <i>CHANNEL</i>	inserted channel group
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC routine channel group software trigger */
adc_software_trigger_enable(ADC_ROUTINE_CHANNEL);
```

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

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

**Table 3-24. Function adc\_end\_of\_conversion\_config**

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

Example:

```
/* configure end of conversion mode */
adc_end_of_conversion_config(ADC_EOC_SET_SEQUENCE);
```

### adc\_resolution\_config

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

**Table 3-25. adc\_resolution\_config**

<b>Function name</b>	adc_resolution_config
<b>Function prototype</b>	void adc_resolution_config(uint32_t resolution);
<b>Function descriptions</b>	configure ADC resolution
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>resolution</b>	configure ADC resolution
ADC_RESOLUTION_12B	12-bit ADC resolution

ADC_RESOLUTION_10B	10-bit ADC resolution
ADC_RESOLUTION_8B	8-bit ADC resolution
ADC_RESOLUTION_6B	6-bit ADC resolution
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the ADC resolution*/
adc_resolution_config(ADC_RESOLUTION_12B);
```

### adc\_routine\_data\_read

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

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

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

Example:

```
/* read ADC routine group 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 3-27. Function adc\_inserted\_data\_read**

Function name	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 group data register
<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 Channelx, x=0,1,2,3
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	ADC conversion value (0~0xFFFF)

Example:

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

```
uint16_t adc_value = 0;
```

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

### adc\_watchdog\_single\_channel\_enable

The description of adc\_watchdog\_single\_channel\_enable is shown as below:

**Table 3-28. Function adc\_watchdog\_single\_channel\_enable**

<b>Function name</b>	adc_watchdog_single_channel_enable
<b>Function prototype</b>	void adc_watchdog_single_channel_enable(uint8_t adc_channel);
<b>Function descriptions</b>	configure ADC analog watchdog 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..10)	ADC channelx(x=0..10)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_watchdog_single_channel_enable(ADC_CHANNEL_1);
```

## adc\_watchdog\_group\_channel\_enable

The description of adc\_watchdog\_group\_channel\_enable is shown as below:

**Table 3-29. Function adc\_watchdog\_group\_channel\_enable**

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

Example:

```
/* configure ADC analog watchdog group channel */
adc_watchdog_group_channel_enable(ADC_ROUTINE_CHANNEL);
```

## adc\_watchdog\_disable

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

**Table 3-30. Function adc\_watchdog\_disable**

<b>Function name</b>	adc_watchdog_disable
<b>Function prototype</b>	void adc_watchdog_disable(void);
<b>Function descriptions</b>	disable ADC analog watchdog
<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 */
```

```
adc_watchdog_disable();
```

### adc\_watchdog\_threshold\_config

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

**Table 3-31. Function adc\_watchdog\_threshold\_config**

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

Example:

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

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

### adc\_oversample\_mode\_config

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

**Table 3-32. 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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

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 3-33. Function adc\_oversample\_mode\_enable**

<b>Function name</b>	adc_oversample_mode_enable
<b>Function prototype</b>	void adc_oversample_mode_enable(void);
<b>Function descriptions</b>	enable 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:

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

```
adc_oversample_mode_enable();
```

### adc\_oversample\_mode\_disable

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

**Table 3-34. 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 3-35. 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_WDE	analog watchdog event flag
ADC_FLAG_EOC	end of group conversion flag
ADC_FLAG_EOIC	end of inserted group conversion flag
ADC_FLAG_STIC	start flag of inserted channel group
ADC_FLAG_STRC	start flag of routine channel group
ADC_FLAG_ROVF	routine data register overflow flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
FlagStatus flag_value;
```

```
flag_value = adc_flag_get(ADC_FLAG_WDE);
```

### adc\_flag\_clear

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

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

<b>Function name</b>	adc_flag_clear
<b>Function prototype</b>	void adc_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear the ADC flag bits
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	the adc flag bits
<i>ADC_FLAG_WDE</i>	analog watchdog event flag
<i>ADC_FLAG_EOC</i>	end of group conversion flag
<i>ADC_FLAG_EOIC</i>	end of inserted group conversion flag
<i>ADC_FLAG_STIC</i>	start flag of inserted channel group
<i>ADC_FLAG_STRC</i>	start flag of routine channel group
<i>ADC_FLAG_ROVF</i>	routine data register overflow flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_flag_clear(ADC_FLAG_WDE);
```

### adc\_interrupt\_enable

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

**Table 3-37. Function adc\_interrupt\_enable**

<b>Function name</b>	adc_interrupt_enable
<b>Function prototype</b>	void adc_interrupt_enable(uint32_t interrupt);
<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
<i>ADC_INT_WDE</i>	analog watchdog interrupt
<i>ADC_INT_EOC</i>	end of group conversion interrupt
<i>ADC_INT_EOIC</i>	end of inserted group conversion interrupt
<i>ADC_INT_ROVF</i>	routine data register overflow interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_interrupt_enable(ADC_INT_WDE);
```

## adc\_interrupt\_disable

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

**Table 3-38. Function adc\_interrupt\_disable**

<b>Function name</b>	adc_interrupt_disable
<b>Function prototype</b>	void adc_interrupt_disable(uint32_t interrupt);
<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
ADC_INT_WDE	analog watchdog interrupt
ADC_INT_EOC	end of group conversion interrupt
ADC_INT_EOIC	end of inserted group conversion interrupt
ADC_INT_ROVF	routine data register overflow interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC analog watchdog interrupt */
adc_interrupt_disable(ADC_INT_WDE);
```

## adc\_interrupt\_flag\_get

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

**Table 3-39. Function adc\_interrupt\_flag\_get**

<b>Function name</b>	adc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus adc_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	get 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_WDE	analog watchdog interrupt
ADC_INT_FLAG_EOC	end of group conversion interrupt
ADC_INT_FLAG_EOIC	end of inserted group conversion interrupt
ADC_INT_FLAG_ROVF	routine data register overflow interrupt
<b>Output parameter{out}</b>	

-	-
Return value	
FlagStatus	SET or RESET

Example:

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

```
FlagStatus flag_value;
```

```
flag_value = adc_interrupt_flag_get(ADC_INT_FLAG_WDE);
```

### adc\_interrupt\_flag\_clear

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

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

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

Example:

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

```
adc_interrupt_flag_clear(ADC_INT_FLAG_WDE);
```

## 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 3-41. CAU Registers**

Registers	Descriptions
CAU_CTL	control register
CAU_STAT0	status register 0
CAU_DI	data input register
CAU_DO	data output register
CAU_DMAEN	DMA enable register
CAU_INTEN	interrupt enable register
CAU_STAT1	status register 1
CAU_INTF	interrupt flag register
CAU_KEY0H	key 0 high register
CAU_KEY0L	key 0 low register
CAU_KEY1H	key 1 high register
CAU_KEY1L	key 1 low register
CAU_KEY2H	key 2 high register
CAU_KEY2L	key 2 low register
CAU_KEY3H	key 3 high register
CAU_KEY3L	key 3 low register
CAU_IV0H	initial vector 0 high register
CAU_IV0L	initial vector 0 low register
CAU_IV1H	initial vector 1 high register
CAU_IV1L	initial vector 1 low register
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 3-42. 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

Function name	Function description
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
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_flag_get	get the CAU flag status
cau_interrupt_enable	enable the CAU interrupts
cau_interrupt_disable	disable the CAU interrupts
cau_interrupt_flag_get	get the interrupt flag

### Structure cau\_key\_parameter\_struct

Table 3-43. 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 3-44. 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 3-45. 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 3-46. 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 3-47. Function cau\_deinit**

<b>Function name</b>	cau_deinit
<b>Function prototype</b>	void cau_deinit(void);
<b>Function descriptions</b>	reset the CAU 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 the CAU peripheral */
```

```
cau_deinit( );
```

### cau\_struct\_para\_init

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

**Table 3-48. 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 3-46. 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 3-49. Function cau\_key\_struct\_para\_init**

<b>Function name</b>	cau_key_struct_para_init
<b>Function prototype</b>	void cau_key_struct_para_init(cau_key_parameter_struct *key_initpara);
<b>Function descriptions</b>	initialize the key parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
key_initpara	the key parameters, refer to structure <a href="#">Table 3-43. Structure cau_key_parameter_struct</a>
<b>Return value</b>	
-	-

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 3-50. Function cau\_iv\_struct\_para\_init**

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

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 3-51. 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>	
<b>cau_context</b>	the context parameter, refer to structure <a href="#">Table 3-45. 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 3-52. 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>	

-	-
Return value	
-	-

Example:

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

```
cau_enable();
```

### cau\_disable

The description of cau\_disable is shown as below:

**Table 3-53. 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 3-54. 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

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
cau_dma_enable(CAU_DMA_INFIFO);
```

### cau\_dma\_disable

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

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

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

Example:

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

```
cau_dma_disable(CAU_DMA_INFIFO);
```

### cau\_init

The description of cau\_init is shown as below:

**Table 3-56. Function cau\_init**

Function name	cau_init
Function prototype	void cau_init(uint32_t alg_dir, uint32_t algo_mode, uint32_t swapping);
Function descriptions	initialize the CAU peripheral
Precondition	-
The called functions	-
Input parameter{in}	

<b>alg_dir</b>	algorithm direction
<i>CAU_ENCRYPT</i>	encrypt
<i>CAU_DECRYPT</i>	decrypt
<b>Input parameter{in}</b>	
<b>algo_mode</b>	algorithm mode selection
<i>CAU_MODE_TDES_ECB</i>	TDES-ECB (3DES Electronic codebook)
<i>CAU_MODE_TDES_CBC</i>	TDES-CBC (3DES Cipher block chaining)
<i>CAU_MODE_DES_ECB</i>	DES-ECB (simple DES Electronic codebook)
<i>CAU_MODE_DES_CBC</i>	DES-CBC (simple DES Cipher block chaining)
<i>CAU_MODE_AES_ECB</i>	AES-ECB (AES Electronic codebook)
<i>CAU_MODE_AES_CBC</i>	AES-CBC (AES Cipher block chaining)
<i>CAU_MODE_AES_CTR</i>	AES-CTR (AES counter mode)
<i>CAU_MODE_AES_KEY</i>	AES decryption key preparation mode
<i>CAU_MODE_AES_GCM</i>	AES-GCM (AES Galois/counter mode)
<i>CAU_MODE_AES_CCM</i>	AES-CCM (AES combined cipher machine mode)
<i>CAU_MODE_AES_CFB</i>	AES-CFB (cipher feedback mode)
<i>CAU_MODE_AES_OFB</i>	AES-OFB (output feedback mode)
<b>Input parameter{in}</b>	
<b>swapping</b>	data swapping selection
<i>CAU_SWAPPING_32BIT</i>	no swapping
<i>CAU_SWAPPING_16BIT</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 3-57. Function cau\_aes\_keysize\_config**

<b>Function name</b>	cau_aes_keysize_config
<b>Function prototype</b>	void cau_aes_keysize_config(uint32_t key_size);
<b>Function descriptions</b>	configure key size if use AES algorithm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>key_size</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-58. Function cau\_key\_init**

<b>Function name</b>	cau_key_init
<b>Function prototype</b>	void cau_key_init(cau_key_parameter_struct* key_initpara);
<b>Function descriptions</b>	initialize the key parameters
<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 3-43. Structure cau_key_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

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 3-59. 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 3-44. 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 3-60. 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
<b>CAU_PREPARE_PHAS</b>	prepare phase

<i>E</i>	
<i>CAU_AAD_PHASE</i>	AAD phase
<i>CAU_ENCRYPT_DEC RYPT_PHASE</i>	encryption/decryption phase
<i>CAU_TAG_PHASE</i>	tag phase
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### cau\_fifo\_flush

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

**Table 3-61. 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 3-62. 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 3-63. 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 3-64. 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>	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	0x0 – 0xFFFFFFFF

Example:

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

```
uint32_t data;
```

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

### cau\_context\_save

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

**Table 3-65. Function cau\_context\_save**

Function name	cau_context_save
Function prototype	void cau_context_save(cau_context_parameter_struct *cau_context, cau_key_parameter_struct* key_initpara)
Function descriptions	save context before context switching
Precondition	-
The called functions	-
Input parameter{in}	
key_initpara	the key parameters, refer to structure <a href="#">Table 3-43. Structure cau_key_parameter_struct</a>
Output parameter{out}	
cau_context	the context, refer to structure <a href="#">Table 3-45. Structure cau_context_parameter_struct</a>
Return value	
-	-

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 3-66. Function cau\_context\_restore**

Function name	cau_context_restore
Function prototype	void cau_context_restore(cau_context_parameter_struct *cau_context);
Function descriptions	restore context after context switching
Precondition	-
The called functions	-
Input parameter{in}	
cau_context	the context, refer to structure <a href="#">Table 3-45. Structure cau_context_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

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 3-67. Function cau\_aes\_ecb**

Function name	cau_aes_ecb
Function prototype	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 3-46.</a> <a href="#">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.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 3-68. 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>	
<b>cau_parameter</b>	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-46. 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 CBC mode */

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

### cau\_aes\_ctr

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

**Table 3-69. 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 3-46.</a> <a href="#">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 3-70. 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>	-

Input parameter{in}	
<b>cau_parameter</b>	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-46.</a> <a href="#">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 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 3-71. 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>	-
Input parameter{in}	
<b>cau_parameter</b>	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-46.</a>

	<a href="#"><u>Structure cau_parameter_struct</u></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 3-72. Function cau\_aes\_gcm**

<b>Function name</b>	cau_aes_gcm
<b>Function prototype</b>	ErrStatus cau_aes_gcm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t *tag)
<b>Function descriptions</b>	encrypt and decrypt using AES in GCM 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="#"><u>Table 3-46. Structure cau_parameter_struct</u></a>
<b>Output parameter{out}</b>	

<b>output</b>	pointer to the returned buffer
<b>Output parameter{out}</b>	
<b>tag</b>	pointer to the returned tag buffer
<b>Return value</b>	
<b>ErrStatus</b>	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 3-73. Function cau\_aes\_ccm**

<b>Function name</b>	cau_aes_ccm
<b>Function prototype</b>	ErrStatus cau_aes_ccm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t tag[], uint32_t tag_size, uint8_t aad_buf[])
<b>Function descriptions</b>	encrypt and decrypt using AES in CCM 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 3-46.</a> <a href="#">Structure cau_parameter_struct</a>
Input parameter{in}	
<b>tag_size</b>	tag size (in bytes)
Output parameter{out}	
<b>output</b>	pointer to the returned buffer
Output parameter{out}	
<b>tag</b>	pointer to the returned tag buffer
Output parameter{out}	
<b>aad_buf</b>	pointer to the user buffer used when formatting aad block
Return value	
<b>ErrStatus</b>	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 3-74. 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>	
<b>cau_parameter</b>	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-46. 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        = 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 3-75. 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>	-
<b>Input parameter{in}</b>	
<b>cau_parameter</b>	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-46.</a> <a href="#">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       = 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 3-76. 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 3-46.</a>

	<a href="#">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 3-77. 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 3-46. 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\_flag\_get

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

**Table 3-78. Function cau\_flag\_get**

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

Return value	
FlagStatus	SET or RESET

Example:

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

```
FlagStatus status;
```

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

### cau\_interrupt\_enable

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

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

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

Example:

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

```
cau_interrupt_enable (CAU_INT_INFIFO);
```

### cau\_interrupt\_disable

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

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

Function name	cau_interrupt_disable
Function prototype	void cau_interrupt_disable(uint32_t interrupt);
Function descriptions	disable the CAU interrupts
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify the CAU interrupt source to be disabled

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 int_flag);
<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);
```

## 3.4. CRC

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

### 3.4.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

**Table 3-82. CRC Registers**

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

### 3.4.2. Descriptions of Peripheral functions

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

**Table 3-83. CRC firmware function**

Function name	Function description
crc_deinit	deinit CRC calculation unit
crc_data_register_reset	reset data register to the value of initializaiton data register
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 a 32-bit data
crc_block_data_calculate	CRC calculate a 32-bit data array

#### crc\_deinit

The description of crc\_deinit is shown as below:

**Table 3-84. Function crc\_deinit**

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

Example:

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

## crc\_data\_register\_reset

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

**Table 3-85. Function crc\_data\_register\_reset**

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

Example:

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

## crc\_data\_register\_read

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

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

<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 value of 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>	
<b>uint32_t</b>	32-bit value of the data register (0-0xFFFFFFFF)

Example:

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

## crc\_free\_data\_register\_read

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

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

<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 value of 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>	
uint8_t	8-bit value of the free data register (0-0xFF)

Example:

```
/* read crc free data register */
uint8_t crc_value = 0;
crc_value = crc_free_data_register_read();
```

## crc\_free\_data\_register\_write

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

**Table 3-88. Function crc\_free\_data\_register\_write**

<b>Function name</b>	crc_free_data_register_write
<b>Function prototype</b>	void crc_free_data_register_write(uint8_t free_data);
<b>Function descriptions</b>	write data to the free data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
free_data	specify 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 3-89. Function crc\_single\_data\_calculate**

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

Example:

```

/* CRC calculate a 32-bit data */

uint32_t val = 0, valcrc = 0;

val = (uint32_t)0xabcd1234;

rcu_periph_clock_enable(RCU_CRC);

valcrc = crc_single_data_calculate(val);

```

## crc\_block\_data\_calculate

The description of crc\_block\_data\_calculate is shown as below:

**Table 3-90. Function crc\_block\_data\_calculate**

<b>Function name</b>	crc_block_data_calculate
<b>Function prototype</b>	uint32_t crc_block_data_calculate(uint32_t array[], uint32_t size);
<b>Function descriptions</b>	calculate the CRC value of an array of 32-bit values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>array</b>	pointer to an array of 32 bits data words
<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>	32-bit CRC calculate value (0-0xFFFFFFFF)

Example:

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

#define BUFFER_SIZE          6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

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

rcu_periph_clock_enable(RCU_CRC);

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

## 3.5. DBG

The DBG hold unit helps debugger to debug power saving mode. The DBG registers are listed in chapter [3.5.1](#). the DBG firmware functions are introduced in chapter [3.5.2](#).

### 3.5.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

**Table 3-91. 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

### 3.5.2. Descriptions of Peripheral functions

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

**Table 3-92. 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

## Enum dbg\_periph\_enum

**Table 3-93. Enum dbg\_periph\_enum**

Member name	Function description
DBG_TIMER1_HOLD	hold TIMER1 counter when core is halted
DBG_TIMER2_HOLD	hold TIMER2 counter when core is halted
DBG_TIMER5_HOLD	hold TIMER5 counter when core is halted
DBG_RTC_HOLD	hold RTC calendar and wakeup counter when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_I2C0_HOLD	hold I2C0 smbus when core is halted
DBG_I2C1_HOLD	hold I2C1 smbus when core is halted
DBG_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 3-94. 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 3-95. Function dbg\_id\_get**

Function name	dbg_id_get
Function prototype	uint32_t dbg_id_get(void);

<b>Function descriptions</b>	read DBG_ID code 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	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 3-96. 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:

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

```
dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

## dbg\_low\_power\_disable

The description of dbg\_low\_power\_disable is shown as below:

**Table 3-97. Function dbg\_low\_power\_disable**

<b>Function name</b>	dbg_low_power_disable
<b>Function prototype</b>	void dbg_low_power_disable(uint32_t dbg_low_power);
<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>dbg_low_power</b>	low power mode
<i>DBG_LOW_POWER_SLEEP</i>	keep debugger connection during sleep mode
<i>DBG_LOW_POWER_DEEPSLEEP</i>	keep debugger connection during deepsleep mode
<i>DBG_LOW_POWER_STANDBY</i>	keep debugger connection during standby mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

## dbg\_periph\_enable

The description of dbg\_periph\_enable is shown as below:

**Table 3-98. Function dbg\_periph\_enable**

<b>Function name</b>	dbg_periph_enable
<b>Function prototype</b>	void dbg_periph_enable(dbg_periph_enum dbg_periph);
<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>dbg_periph</b>	Peripheral refer to <a href="#">Table 3-93. Enum dbg_periph_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### dbg\_periph\_disable

The description of dbg\_periph\_disable is shown as below:

**Table 3-99. Function dbg\_periph\_disable**

<b>Function name</b>	dbg_periph_disable
<b>Function prototype</b>	void dbg_periph_disable(dbg_periph_enum dbg_periph);
<b>Function descriptions</b>	disable peripheral behavior when the mcu is in debug mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dbg_periph</b>	peripheral refer to <a href="#">Table 3-93. Enum dbg_periph_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### 3.6.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

**Table 3-100. DMA Registers**

Registers	Descriptions
DMA_INTF0	Interrupt flag register 0
DMA_INTF1	Interrupt flag register 1
DMA_INTC0	Interrupt flag clear register 0
DMA_INTC1	Interrupt flag clear register 1

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

### 3.6.2. Descriptions of Peripheral functions

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

**Table 3-101. 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

Function name	Function description
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
dma_using_memory_get	get DMA using memory
dma_channel_subperipheral_select	select DMA channel peripheral
dma_flow_controller_config	configure DMA flow controller
dma_switch_buffer_mode_enable	enable DMA switch buffer mode
dma_switch_buffer_mode_disable	disable DMA switch buffer mode
dma_fifo_status_get	get DMA FIFO status
dma_flag_get	check the designated flag of a DMA channel is set or not
dma_flag_clear	clear the designated flag of a DMA channel
dma_interrupt_enable	enable DMA interrupt
dma_interrupt_disable	disable DMA interrupt
dma_interrupt_flag_get	check the designated interrupt flag of a DMA channel is set or not
dma_interrupt_flag_clear	clear the designated interrupt flag of a DMA channel

### Enum dma\_channel\_enum

**Table 3-102. Enum dma\_channel\_enum**

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

### Enum dma\_subperipheral\_enum

**Table 3-103. Enum dma\_subperipheral\_enum**

Member name	Function description
DMA_SUBPERI0	DMA Peripheral 0
DMA_SUBPERI1	DMA Peripheral 1
DMA_SUBPERI2	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 3-104. 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 3-105. 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 3-106. Function dma\_deinit**

Function name	dma_deinit
Function prototype	void dma_deinit(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>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* DMA channel0 deinitialize*/
dma_deinit(DMA_CH0);
```

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

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

**Table 3-107. 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>Output parameter{out}</b>	
<b>init_struct</b>	the address of structure for initialization, the structure members can refer to <a href="#">Table 3-105. Structure dma_single_data_parameter_struct</a>
<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 3-108. Function dma\_multi\_data\_para\_struct\_init

Function name	dma_multi_data_para_struct_init
Function prototype	void dma_multi_data_para_struct_init(dma_multi_data_parameter_struct* init_struct);
Function descriptions	initialize the DMA multi data mode parameters struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_struct	the address of structure for initialization, the structure members can refer to <a href="#">Table 3-104. Structure dma_multi_data_parameter_struct</a>
Return value	
-	-

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 3-109. Function dma\_single\_data\_mode\_init

Function name	dma_single_data_mode_init
Function prototype	void dma_single_data_mode_init(dma_channel_enum channelx, dma_single_data_parameter_struct* init_struct);
Function descriptions	initialize DMA single data mode
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..7)	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
Input parameter{in}	
init_struct	the address of structure for initialization, the structure members can refer to <a href="#">Table 3-105. Structure dma_single_data_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* DMA channel0 single data mode initialize */

dma_single_data_parameter_struct dma_init_struct;

dma_deinit(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 3-110. Function dma\_multi\_data\_mode\_init**

Function name	dma_multi_data_mode_init
Function prototype	void dma_multi_data_mode_init(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}	
channelx	DMA channel
DMA_CHx(x=0..7)	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
Input parameter{in}	
init_struct	the address of structure for initialization, the structure members can refer to <a href="#">Table 3-104. Structure dma_multi_data_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* DMA channel0 multi data mode initialize */

dma_multi_data_parameter_struct dma_init_parameter;

dma_multi_data_para_struct_init(&dma_init_parameter);

dma_deinit(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 3-111. Function dma\_periph\_address\_config**

Function name	dma_periph_address_config
Function prototype	void dma_periph_address_config(dma_channel_enum channelx, uint32_t address);
Function descriptions	set DMA peripheral base address
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>

Input parameter{in}	
address	peripheral base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 peripheral address */

#define USART_DATA_ADDR (USART0 + 0x00000024U)

dma_periph_address_config (DMA_CH0, USART_DATA_ADDR);
```

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

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

**Table 3-112. Function dma\_memory\_address\_config**

Function name	dma_memory_address_config
Function prototype	void dma_memory_address_config(dma_channel_enum channelx, uint8_t memory_flag, uint32_t address);
Function descriptions	set DMA memory0 base address
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
Input parameter{in}	
address	memory0 base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DMA channel0 memory0 address */

uint32_t array[10] = {0};

dma_memory_address_config (DMA_CH0, array);
```

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

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

Table 3-113. Function dma\_transfer\_number\_config

<b>Function name</b>	dma_transfer_number_config
<b>Function prototype</b>	void dma_transfer_number_config(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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. 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 DMA channel0 transfer number */
#define TRANSFER_NUM                0x400
dma_transfer_number_config(DMA_CH0, TRANSFER_NUM);
```

### dma\_transfer\_number\_get

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

Table 3-114. Function dma\_transfer\_number\_get

<b>Function name</b>	dma_transfer_number_get
<b>Function prototype</b>	uint32_t dma_transfer_number_get(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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	0x00000000 – 0x0000FFFF

Example:

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

```
uint32_t number = 0;
```

```
number = dma_transfer_number_get(DMA_CH0);
```

### **dma\_priority\_config**

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

**Table 3-115. Function dma\_priority\_config**

<b>Function name</b>	dma_priority_config
<b>Function prototype</b>	void dma_priority_config(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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>priority</b>	priority Level of this channel
<i>DMA_PRIORITY_LOW</i>	low priority
<i>DMA_PRIORITY_MEDIUM</i>	medium priority
<i>DMA_PRIORITY_HIGH</i>	high priority
<i>DMA_PRIORITY_ULTRA_HIGH</i>	ultra high priority
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA channel0 priority */
```

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

### **dma\_memory\_burst\_beats\_config**

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

**Table 3-116. 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(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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. 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 DMA channel0 transfer burst beats of memory */
```

```
dma_memory_burst_beats_config(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 3-117. 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 (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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. 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 DMA channel0 transfer burst beats of peripheral */
dma_periph_burst_beats_config(DMA_CH0, DMA_PERIPH_BURST_8_BEAT);
```

### **dma\_memory\_width\_config**

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

**Table 3-118. Function dma\_memory\_width\_config**

<b>Function name</b>	dma_memory_width_config
<b>Function prototype</b>	void dma_memory_width_config(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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>msize</b>	transfer data width of memory
<i>DMA_MEMORY_WIDTH_H_8BIT</i>	transfer data width of memory is 8-bit
<i>DMA_MEMORY_WIDTH_H_16BIT</i>	transfer data width of memory is 16-bit
<i>DMA_MEMORY_WIDTH_H_32BIT</i>	transfer data width of memory is 32-bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA channel0 transfer memory width */
dma_memory_width_config(DMA_CH0, DMA_MEMORY_WIDTH_8BIT);
```

## dma\_periph\_width\_config

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

**Table 3-119. Function dma\_periph\_width\_config**

<b>Function name</b>	dma_periph_width_config
<b>Function prototype</b>	void dma_periph_width_config (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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>psize</b>	transfer data width of peripheral
<i>DMA_PERIPHERAL_WIDTH_8BIT</i>	transfer data width of peripheral is 8-bit
<i>DMA_PERIPHERAL_WIDTH_16BIT</i>	transfer data width of peripheral is 16-bit
<i>DMA_PERIPHERAL_WIDTH_32BIT</i>	transfer data width of peripheral is 32-bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## dma\_memory\_address\_generation\_config

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

**Table 3-120. 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(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>channelx</b>	DMA channel

<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. 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 DMA channel0 memory address generation algorithm as memory increase*/
```

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

### **dma\_peripheral\_address\_generation\_config**

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

**Table 3-121. 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(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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. 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 DMA channel0 peripheral address generation algorithm as peripheral increase*/
```

```
dma_peripheral_address_generation_config(DMA_CH0, DMA_PERIPH_INCREASE_ENABLE);
```

## dma\_circulation\_enable

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

**Table 3-122. Function dma\_circulation\_enable**

<b>Function name</b>	dma_circulation_enable
<b>Function prototype</b>	void dma_circulation_enable(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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA channel0 circulation mode */
```

```
dma_circulation_enable(DMA_CH0);
```

## dma\_circulation\_disable

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

**Table 3-123. Function dma\_circulation\_disable**

<b>Function name</b>	dma_circulation_disable
<b>Function prototype</b>	void dma_circulation_disable(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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA channel0 circulation mode */
```

```
dma_circulation_disable(DMA_CH0);
```

## dma\_channel\_enable

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

**Table 3-124. Function dma\_channel\_enable**

<b>Function name</b>	dma_channel_enable
<b>Function prototype</b>	void dma_channel_enable(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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA channel0 */
```

```
dma_channel_enable(DMA_CH0);
```

## dma\_channel\_disable

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

**Table 3-125. Function dma\_channel\_disable**

<b>Function name</b>	dma_channel_disable
<b>Function prototype</b>	void dma_channel_disable(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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA channel0 */
```

```
dma_channel_disable(DMA_CH0);
```

### dma\_transfer\_direction\_config

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

**Table 3-126. Function dma\_transfer\_direction\_config**

<b>Function name</b>	dma_transfer_direction_config
<b>Function prototype</b>	void dma_transfer_direction_config(dma_channel_enum channelx, uint8_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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. 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 DMA channel0 transfer direction as peripheral to memory mode*/
```

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

### dma\_switch\_buffer\_mode\_config

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

**Table 3-127. 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(dma_channel_enum channelx, uint32_t memory1_addr, uint32_t memory_select);
<b>Function descriptions</b>	configure DMA switch buffer mode
<b>Precondition</b>	-

The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
Input parameter{in}	
memory1_addr	memory1 base address
Input parameter{in}	
memory_select	select memory transfer area
DMA_MEMORY_0	select memory0 as memory transfer area
DMA_MEMORY_1	select memory1 as memory transfer area
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

uint32_t mem[32] = {0};

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

### dma\_using\_memory\_get

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

**Table 3-128. Function dma\_using\_memory\_get**

Function name	dma_using_memory_get
Function prototype	uint32_t dma_using_memory_get(dma_channel_enum channelx);
Function descriptions	configure DMA switch buffer mode
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
Output parameter{out}	
-	-
Return value	
uint32_t	the memory buffer area currently used
DMA_MEMORY_0	memory0 is selected as memory transfer area
DMA_MEMORY_1	memory1 is selected as memory transfer area

Example:

```
/* get DMA channel0 transfer buffer currently used */
```

```
uint32_t buf_addr = DMA_MEMORY_0;
```

```
buf_addr = dma_using_memory_get(DMA_CH0);
```

## dma\_channel\_subperipheral\_select

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

**Table 3-129. Function dma\_channel\_subperipheral\_select**

<b>Function name</b>	dma_channel_subperipheral_select
<b>Function prototype</b>	void dma_channel_subperipheral_select(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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. 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)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA channel0 peripheral */
```

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

## dma\_flow\_controller\_config

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

**Table 3-130. Function dma\_flow\_controller\_config**

<b>Function name</b>	dma_flow_controller_config
<b>Function prototype</b>	void dma_flow_controller_config(dma_channel_enum channelx, uint32_t controller);
<b>Function descriptions</b>	configure DMA flow controller
<b>Precondition</b>	-
<b>The called functions</b>	-
<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 3-102. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA channel0 flow controller */
dma_flow_controller_config(DMA_CH0, DMA_FLOW_CONTROLLER_DMA);
```

### **dma\_switch\_buffer\_mode\_enable**

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

**Table 3-131. 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(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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA channel0 switch buffer mode */
dma_switch_buffer_mode_enable (DMA_CH0);
```

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

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

Table 3-132. Function dma\_switch\_buffer\_mode\_enable

Function name	dma_switch_buffer_mode_disable
Function prototype	void dma_switch_buffer_mode_disable(dma_channel_enum channelx);
Function descriptions	enable DMA switch buffer mode
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA channel0 switch buffer mode */
dma_switch_buffer_mode_disable (DMA_CH0);
```

### dma\_fifo\_status\_get

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

Table 3-133. Function dma\_fifo\_status\_get

Function name	dma_fifo_status_get
Function prototype	uint32_t dma_fifo_status_get(dma_channel_enum channelx);
Function descriptions	get DMA FIFO status
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0...7)	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
Output parameter{out}	
-	-
Return value	
uint32_t	the using memory in FIFO

Example:

```
/* get DMA channel0 FIFO status */
uint32_t fifo_state = 0;
fifo_state = dma_fifo_status_get(DMA_CH0);
```

## dma\_flag\_get

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

**Table 3-134. Function dma\_flag\_get**

<b>Function name</b>	dma_flag_get
<b>Function prototype</b>	FlagStatus dma_flag_get(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>	-
<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 3-102. 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>	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
FlagStatus state = RESET;
```

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

## dma\_flag\_clear

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

**Table 3-135. Function dma\_flag\_clear**

<b>Function name</b>	dma_flag_clear
<b>Function prototype</b>	void dma_flag_clear(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>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. 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	
-	-

Example:

```
/* clear DMA channel0 full transfer finish flag */
dma_flag_clear(DMA_CH0, DMA_FLAG_FTF);
```

### **dma\_interrupt\_enable**

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

**Table 3-136. Function dma\_interrupt\_enable**

<b>Function name</b>	dma_interrupt_enable
<b>Function prototype</b>	void dma_interrupt_enable(dma_channel_enum channelx, uint32_t interrupt);
<b>Function descriptions</b>	enable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0...7)</i>	DMA channel selection, refer to <a href="#">Table 3-102. Enum dma_channel_enum</a>
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### dma\_interrupt\_disable

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

**Table 3-137. Function dma\_interrupt\_disable**

<b>Function name</b>	dma_interrupt_disable
<b>Function prototype</b>	void dma_interrupt_disable(dma_channel_enum channelx, uint32_t interrupt);
<b>Function descriptions</b>	disable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<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 3-102. 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 DMA channel0 full transfer finish interrupt */
```

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

### dma\_interrupt\_flag\_get

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

**Table 3-138. Function dma\_interrupt\_flag\_get**

<b>Function name</b>	dma_interrupt_flag_get
<b>Function prototype</b>	FlagStatus dma_interrupt_flag_get(dma_channel_enum channelx, uint32_t int_flag);
<b>Function descriptions</b>	check the designated interrupt flag of a DMA channel is set or not
<b>Precondition</b>	-

<b>The called functions</b>	-
<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 3-102. 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>	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
FlagStatus state = RESET;
```

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

### **dma\_interrupt\_flag\_clear**

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

**Table 3-139. Function dma\_interrupt\_flag\_clear**

<b>Function name</b>	dma_interrupt_flag_clear
<b>Function prototype</b>	void dma_interrupt_flag_clear(dma_channel_enum channelx, uint32_t int_flag);
<b>Function descriptions</b>	clear the designated interrupt flag of a DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<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 3-102. 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

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear DMA channel0 full transfer finish interrupt flag */
```

```
dma_interrupt_flag_clear(DMA_CH0, DMA_INT_FLAG_FTF);
```

## 3.7. ECLIC

RISC-V integrates the Enhancement Core-Local Interrupt Controller (ECLIC) for efficient interrupts processing. ECLIC is designed to provide low-latency, vectored, pre-emptive interrupts for RISC-V systems. The ECLIC firmware functions are introduced in chapter [3.7.1](#).

### 3.7.1. Descriptions of Peripheral functions

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

**Table 3-140. ECLIC firmware function**

Function name	Function description
eclic_global_interrupt_enable	enable the global interrupt
eclic_global_interrupt_disable	disable the global interrupt
eclic_level_threshold_set	set machine mode interrupt level threshold
eclic_priority_group_set	set the priority group
eclic_irq_enable	enable the interrupt request
eclic_irq_disable	disable the interrupt request
eclic_system_reset	reset system

### Enum IRQn\_Type

**Table3-141. Enum IRQn\_Type**

Member name	Function description
CLIC_INT_SFT	Software interrupt
CLIC_INT_TMR	CPU Timer interrupt
WWDGT_IRQn	window watchDog timer interrupt
LVD_IRQn	LVD through EXTI line detect interrupt
TAMPER_STAMP_IRQn	tamper and timestamp through EXTI line detect
RTC_WKUP_IRQn	RTC wakeup through EXTI line interrupt
FMC_IRQn	FMC interrupt
RCU_IRQn	RCU interrupt
EXTI0_IRQn	EXTI line 0 interrupts

Member name	Function description
CLIC_INT_SFT	Software interrupt
CLIC_INT_TMR	CPU Timer interrupt
EXTI1_IRQn	EXTI line 1 interrupts
EXTI2_IRQn	EXTI line 2 interrupts
EXTI3_IRQn	EXTI line 3 interrupts
EXTI4_IRQn	EXTI line 4 interrupts
DMA_Channel0_IRQn	DMA0 channel0 interrupt
DMA_Channel1_IRQn	DMA0 channel1 interrupt
DMA_Channel2_IRQn	DMA0 channel2 interrupt
DMA_Channel3_IRQn	DMA0 channel3 interrupt
DMA_Channel4_IRQn	DMA0 channel4 interrupt
DMA_Channel5_IRQn	DMA0 channel5 interrupt
DMA_Channel6_IRQn	DMA0 channel6 interrupt
DMA_Channel7_IRQn	DMA0 channel7 interrupt
ADC_IRQn	ADC interrupt
EXTI5_9_IRQn	EXTI[9:5] interrupts
TIMER0_BRK_IRQn	TIMER0 break interrupts
TIMER0_UP_IRQn	TIMER0 update interrupts
TIMER0_CMT_IRQn	TIMER0 commutation interrupts
TIMER0_Channel_IRQn	TIMER0 channel capture compare interrupts
TIMER1_IRQn	TIMER1 interrupt
TIMER2_IRQn	TIMER2 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
SPI_IRQn	SPI interrupt
USART0_IRQn	USART0 interrupt
USART1_IRQn	USART1 interrupt
USART2_IRQn	USART2 interrupt
EXTI10_15_IRQn	EXTI[15:10] interrupts
RTC_Alarm_IRQn	RTC alarm interrupt
VLVDF_IRQn	VLVDF interrupt
TIMER15_IRQn	TIMER15 global interrupt
TIMER16_IRQn	TIMER16 global interrupt
I2C0_WKUP_IRQn	I2C0 wakeup interrupt
USART0_WKUP_IRQn	USART0 wakeup interrupt
TIMER5_IRQn	TIMER5 global interrupt
WIFI_TRIGGER_IRQn	WIFI protocol trigger interrupt
WIFI_MAC_IRQn	WIFI MAC interrupt
WIFI_TX_IRQn	WIFI Tx interrupt

Member name	Function description
CLIC_INT_SFT	Software interrupt
CLIC_INT_TMR	CPU Timer interrupt
WIFI_RX_IRQn	WIFI Rx interrupt
LA_IRQn	LA interrupt
WIFI_WKUP_IRQn	WIFI wakeup interrupt
BLE_WKUP_IRQn	BLE wakeup interrupt
PLATFORM_WAKE_IRQn	Platform wake interrupt
ISO_BT_STAMP0_IRQn	ISO Bluetooth TimeStamp interrupt0
ISO_BT_STAMP1_IRQn	ISO Bluetooth TimeStamp interrupt1
ISO_BT_STAMP2_IRQn	ISO Bluetooth TimeStamp interrupt2
ISO_BT_STAMP3_IRQn	ISO Bluetooth TimeStamp interrupt3
ISO_BT_STAMP4_IRQn	ISO Bluetooth TimeStamp interrupt4
ISO_BT_STAMP5_IRQn	ISO Bluetooth TimeStamp interrupt5
ISO_BT_STAMP6_IRQn	ISO Bluetooth TimeStamp interrupt6
ISO_BT_STAMP7_IRQn	ISO Bluetooth TimeStamp interrupt7
PMU_IRQn	PMU interrupt
CAU_IRQn	CAU interrupt
HAU_TRNG_IRQn	HAU and TRNG interrupt
WIFI_INT_IRQn	WIFI global interrupt
WIFI_SW_TRIG_IRQn	SW triggered interrupt
WIFI_FINE_TIMER_TARGET_IRQn	Fine timer target interrupt
WIFI_STAMP_TARGET1_IRQn	Timestamp target 1 interrupt
WIFI_STAMP_TARGET2_IRQn	Timestamp target 2 interrupt
WIFI_STAMP_TARGET3_IRQn	Timestamp target 3 interrupt
WIFI_ENCRYPTION_ENGINE_IRQn	Encryption engine interrupt
WIFI_SLEEP_MODE_IRQn	Sleep mode interrupt
WIFI_HALF_SLOT_IRQn	Half slot interrupt
WIFI_FIFO_ACTIVITY_IRQn	FIFO activity interrupt
WIFI_ERROR_IRQn	Error interrupt
WIFI_FREQ_SELECT_IRQn	Frequency selection interrupt
EFUSE_IRQn	EFUSE global interrupt
QSPI_IRQn	QSPI global interrupt
PKCAU_IRQn	PKCAU global interrupt

### eclic\_global\_interrupt\_enable

The description of eclic\_global\_interrupt\_enable is shown as below:

**Table 3-142. Function eclic\_global\_interrupt\_enable**

Function name	eclic_global_interrupt_enable
Function prototype	void eclic_global_interrupt_enable(void);
Function descriptions	enable the global interrupt

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

Example:

```
/* enable the global interrupt */
eclic_global_interrupt_enable();
```

### eclic\_global\_interrupt\_disable

The description of eclic\_global\_interrupt\_disable is shown as below:

**Table 3-143. Function eclic\_global\_interrupt\_disable**

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

Example:

```
/* disable the global interrupt */
eclic_global_interrupt_disable();
```

### eclic\_level\_threshold\_set

The description of eclic\_level\_threshold\_set is shown as below:

**Table 3-144. Function eclic\_level\_threshold\_set**

Function name	eclic_level_threshold_set
Function prototype	void eclic_level_threshold_set(uint8_t threshold);
Function descriptions	set machine mode interrupt level threshold
Precondition	-
The called functions	-

Input parameter{in}	
<b>threshold</b>	the level threshold needed to set (0 ~ 15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set machine mode interrupt level threshold */
eclic_level_threshold_set(0);
```

### eclic\_priority\_group\_set

The description of eclic\_priority\_group\_set is shown as below:

**Table 3-145. Function eclic\_priority\_group\_set**

<b>Function name</b>	eclic_priority_group_set
<b>Function prototype</b>	void eclic_priority_group_set(uint8_t prigroup);
<b>Function descriptions</b>	set the priority group
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>prigroup</b>	specify the priority group
<i>ECLIC_PRIGROUP_LEV EL0_PRIO4</i>	0 bits for level, 4 bits for priority
<i>ECLIC_PRIGROUP_LEV EL1_PRIO3</i>	1 bits for level, 3 bits for priority
<i>ECLIC_PRIGROUP_LEV EL2_PRIO2</i>	2 bits for level, 2 bits for priority
<i>ECLIC_PRIGROUP_LEV EL3_PRIO1</i>	3 bits for level, 1 bits for priority
<i>ECLIC_PRIGROUP_LEV EL4_PRIO0</i>	4 bits for level, 0 bits for priority
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the priority group */
eclic_priority_group_set(ECLIC_PRIGROUP_LEVEL0_PRIO4);
```

## eclic\_irq\_enable

The description of eclic\_irq\_enable is shown as below:

**Table 3-146. Function eclic\_irq\_enable**

<b>Function name</b>	eclic_irq_enable
<b>Function prototype</b>	void eclic_irq_enable(IRQn_Type source, uint8_t level, uint8_t priority);
<b>Function descriptions</b>	enable the interrupt request
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	interrupt request, detailed in <a href="#">Table3-141. Enum IRQn_Type</a>
<b>Input parameter{in}</b>	
<b>level</b>	the level needed to set (maximum is 15, refer to the priority group)
<b>Input parameter{in}</b>	
<b>priority</b>	the priority needed to set (maximum is 15, refer to the priority group)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable and set key EXTI interrupt to the specified priority */
eclic_global_interrupt_enable();
eclic_priority_group_set(ECLIC_PRIGROUP_LEVEL3_PRIO1);
eclic_irq_enable(EXTI10_15_IRQn, 1, 1);
```

## eclic\_irq\_disable

The description of eclic\_irq\_disable is shown as below:

**Table 3-147. Function eclic\_irq\_disable**

<b>Function name</b>	eclic_irq_disable
<b>Function prototype</b>	void eclic_irq_disable(IRQn_Type source)
<b>Function descriptions</b>	disable the interrupt request
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	interrupt request, detailed in <a href="#">Table3-141. Enum IRQn_Type</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the interrupt request */
eclic_irq_disable(EXTI10_15_IRQn);
```

## eclic\_system\_reset

The description of eclic\_system\_reset is shown as below:

**Table 3-148. Function eclic\_system\_reset**

<b>Function name</b>	eclic_system_reset
<b>Function prototype</b>	void eclic_system_reset(void);
<b>Function descriptions</b>	reset system
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### 3.8.1. Descriptions of Peripheral registers

EFUSE registers are listed in the table shown as below:

**Table 3-149. EFUSE Registers**

Registers	Descriptions
EFUSE_CS	control and status register
EFUSE_ADDR	address register
EFUSE_CTL0	control 0 register
EFUSE_CTL1	control 1 register
EFUSE_FPCTL	flash protection control register
EFUSE_USERCTL	user control register
EFUSE_RESx(x =	mcu reserved register x(x = 0...2)

Registers	Descriptions
0...2)	
EFUSE_AESKEYx( x = 0...3)	firmware AES key register x(x = 0...3)
EFUSE_ROTTPKKE Yx(x = 0...7)	RoTPK key register x(x = 0...7)
EFUSE_PUIDx(x = 0...3)	product UID register x(x = 0...3)
EFUSE_HUKKEYx( x = 0...3)	HUK key register x(x = 0...3)
EFUSE_USER_DA TAX(x = 0...7)	user data register x(x = 0...7)
EFUSE_BOOTADD R	boot address register

### 3.8.2. Descriptions of Peripheral functions

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

**Table 3-150. EFUSE firmware function**

Function name	Function description
efuse_read	read EFUSE value
efuse_write	write EFUSE
efuse_boot_config	configure boot field
efuse_control1_config	configure efuse control1 field
efuse_fp_config	configure flash protection field
efuse_user_control_config	configure user control
efuse_res_write	write MCU reserved value
efuse_aes_key_write	write AES key
efuse_rotpk_key_write	write ROTPK key
efuse_user_data_write	write user data
efuse_res_read	read MCU reserved value
efuse_aes_key_read	read AES key
efuse_rotpk_key_read	read ROTPK key
efuse_puid_read	read puid
efuse_huk_key_read	read huk key
efuse_user_data_read	read user data
efuse_boot_address_get	get boot address information
efuse_lock_config	lock efuse filed
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

Function name	Function description
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 3-151. Enum efuse\_flag\_enum

Member name	Function description
EFUSE_PGIF	programming operation completion flag
EFUSE_RDIF	read operation completion flag
EFUSE_OVBERIF	overstep boundary error flag

### Enum efuse\_clear\_flag\_enum

Table 3-152. Enum efuse\_clear\_flag\_enum

Member name	Function description
EFUSE_PGIC	clear programming operation completion flag
EFUSE_RDIC	clear read operation completion flag
EFUSE_OVBERIC	clear overstep boundary error flag

### Enum efuse\_int\_enum

Table 3-153. Enum efuse\_int\_enum

Member name	Function description
EFUSE_INTEN_PG	programming operation completion interrupt
EFUSE_INTEN_RD	read operation completion interrupt
EFUSE_INTEN_OVBER R	overstep boundary error interrupt

### Enum efuse\_int\_flag\_enum

Table 3-154. 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_OVBERIF	overstep boundary error interrupt flag

### Enum efuse\_clear\_int\_flag\_enum

Table 3-155. 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_OVBERIC	clear overstep boundary error interrupt flag

## Enum efuse\_reg\_lock\_enum

**Table 3-156. Enum efuse\_reg\_lock\_enum**

Member name	Function description
EFUSE_BOOT_LOCK	EFUSE boot bits lock bit
EFUSE_ROTPKKEY_LOCK	EFUSE_ROTPKKEY register lock bit
EFUSE_USER_DATA_LOCK	EFUSE_USER_DATA register lock bit
EFUSE_AESKEY_LOCK	EFUSE_AESKEY register lock bit
EFUSE_FPCTL_USERCTL_LOCK	EFUSE_FPCTL and EFUSE_USERCTL register lock bit

## efuse\_read

The description of efuse\_read is shown as below:

**Table 3-157. 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 – 0x60)
Input parameter{in}	
size	size of EFUSE (0x01 – 0x20 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(0x60, 32, buffer);

```

## efuse\_write

The description of efuse\_write is shown as below:

Table 3-158. 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 – 0x60), 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 – 0x20 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(0x60, 8, buffer);
```

### efuse\_boot\_config

The description of efuse\_boot\_config is shown as below:

Table 3-159. 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>	configure boot
<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
<b>Input parameter{in}</b>	
<b>bt_value</b>	the value of boot configuration
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<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\_control1\_config

The description of efuse\_control1\_config is shown as below:

**Table 3-160. Function efuse\_control1\_config**

<b>Function name</b>	efuse_control1_config
<b>Function prototype</b>	ErrStatus efuse_control1_config(uint32_t size, uint8_t ctl[]);
<b>Function descriptions</b>	configure efuse control1
<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
<b>Input parameter{in}</b>	
<b>ctl</b>	control1 value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* write control1 value */
```

```
uint8_t ctl[1] = 0x02;
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_control1_config(1, ctl);
```

### efuse\_fp\_config

The description of efuse\_fp\_config is shown as below:

**Table 3-161. Function efuse\_fp\_config**

<b>Function name</b>	efuse_fp_config
<b>Function prototype</b>	ErrStatus efuse_fp_config(uint32_t size, uint8_t fp_value[]);
<b>Function descriptions</b>	flash protection 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>fp_value</b>	flash protection value
Output parameter{out}	
-	-
Return value	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* write Flash protection value */

uint8_t fp_value[1] = 0x01;

ErrStatus flag = ERROR;

flag = efuse_fp_config(1, fp_value);
```

### efuse\_user\_control\_config

The description of efuse\_user\_control\_config is shown as below:

**Table 3-162. Function efuse\_user\_control\_config**

<b>Function name</b>	efuse_user_control_config
<b>Function prototype</b>	ErrStatus efuse_user_control_config(uint32_t size, uint8_t user_ctl[]);
<b>Function descriptions</b>	configure user control
<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>user_ctl</b>	user control value
Output parameter{out}	
-	-
Return value	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* configure user control */

uint8_t user_ctl[1] = 0x02;

ErrStatus flag = ERROR;

flag = efuse_user_control_config(1, user_ctl);
```

### efuse\_res\_write

The description of efuse\_res\_write is shown as below:

Table 3-163. Function efuse\_res\_write

Function name	efuse_res_write
Function prototype	ErrStatus efuse_res_write(uint32_t size, uint8_t buf[]);
Function descriptions	write MCU reserved value
Precondition	-
The called functions	efuse_write
Input parameter{in}	
size	size of data (byte), the size must be 12
Input parameter{in}	
buf	MCU reserved value
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* write MCU reserved value */
```

```
uint32_t buffer[3] = {0x11223344, 0x55667788, 0x9900aabb};
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_res_write(12, (uint8_t *)buffer);
```

### efuse\_aes\_key\_write

The description of efuse\_aes\_key\_write is shown as below:

Table 3-164. Function efuse\_aes\_key\_write

Function name	efuse_aes_key_write
Function prototype	ErrStatus efuse_aes_key_write(uint32_t size, uint8_t buf[]);
Function descriptions	write AES key
Precondition	-
The called functions	efuse_write
Input parameter{in}	
size	size of data (byte), the size must be 16
Input parameter{in}	
buf	the buffer of data written to efuse
Output parameter{out}	
-	-
Return value	
ErrStatus	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 3-165. Function efuse\_rotpk\_key\_write**

<b>Function name</b>	efuse_rotpk_key_write
<b>Function prototype</b>	ErrStatus efuse_rotpk_key_write(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\_user\_data\_write

The description of efuse\_user\_data\_write is shown as below:

**Table 3-166. Function efuse\_user\_data\_write**

<b>Function name</b>	efuse_user_data_write
<b>Function prototype</b>	ErrStatus efuse_user_data_write(uint32_t size, uint8_t buf[]);
<b>Function descriptions</b>	write user data
<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

Input parameter{in}	
<b>buf</b>	the buffer of data written to efuse
Output parameter{out}	
-	-
Return value	
<b>ErrStatus</b>	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\_res\_read

The description of efuse\_res\_read is shown as below:

**Table 3-167. Function efuse\_res\_read**

<b>Function name</b>	efuse_res_read
<b>Function prototype</b>	void efuse_res_read(uint32_t buf[]);
<b>Function descriptions</b>	read MCU reserved value
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
<b>buf</b>	MCU reserved value after system reset
Return value	
-	-

Example:

```
/* read MCU reserved value */

uint32_t buffer[3];

efuse_res_read(*buffer);
```

### efuse\_aes\_key\_read

The description of efuse\_aes\_key\_read is shown as below:

**Table 3-168. Function efuse\_aes\_key\_read**

<b>Function name</b>	efuse_aes_key_read
----------------------	--------------------

<b>Function prototype</b>	void efuse_aes_key_read(uint32_t buf[]);
<b>Function descriptions</b>	read AES key
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
buf	AES_KEY data after system reset
<b>Return value</b>	
-	-

Example:

```
/* read AES key */
uint32_t buffer[4];
efuse_aes_key_read(*buffer);
```

### efuse\_rotpk\_key\_read

The description of efuse\_rotpk\_key\_read is shown as below:

**Table 3-169. Function efuse\_rotpk\_key\_read**

<b>Function name</b>	efuse_rotpk_key_read
<b>Function prototype</b>	void efuse_rotpk_key_read(uint32_t size, uint32_t buf[]);
<b>Function descriptions</b>	read ROTPK key
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
buf	rotpk_key data after system reset
<b>Return value</b>	
-	-

Example:

```
/* read ROTPK key */
uint32_t buffer[8];
efuse_rotpk_key_read(*buffer);
```

### efuse\_puid\_read

The description of efuse\_puid\_read is shown as below:

Table 3-170. Function efuse\_puid\_read

Function name	efuse_puid_read
Function prototype	void efuse_puid_read(uint32_t buf[]);
Function descriptions	read puid
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
buf	puid data after system reset
Return value	
-	-

Example:

```
/* read puid */
uint32_t buffer[4];
efuse_puid_read(*buffer);
```

### efuse\_huk\_key\_read

The description of efuse\_huk\_key\_read is shown as below:

Table 3-171. Function efuse\_huk\_key\_read

Function name	efuse_huk_key_read
Function prototype	void efuse_huk_key_read(uint32_t buf[]);
Function descriptions	read huk key
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
buf	huk key data after system reset
Return value	
-	-

Example:

```
/* read huk key */
uint32_t buffer[4];
efuse_huk_key_read(*buffer);
```

## efuse\_user\_data\_read

The description of efuse\_user\_data\_read is shown as below:

**Table 3-172. Function efuse\_user\_data\_read**

<b>Function name</b>	efuse_user_data_read
<b>Function prototype</b>	void efuse_user_data_read(uint32_t buf[]);
<b>Function descriptions</b>	read user data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
buf	user data after system reset
<b>Return value</b>	
-	-

Example:

```
/* read user data */
uint32_t buffer[8];
efuse_user_data_read(*buffer);
```

## efuse\_boot\_address\_get

The description of efuse\_boot\_address\_get is shown as below:

**Table 3-173. Function efuse\_boot\_address\_get**

<b>Function name</b>	efuse_boot_address_get
<b>Function prototype</b>	uint32_t efuse_boot_address_get(void);
<b>Function descriptions</b>	get boot address information
<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 boot address (0x0 – 0xFFFFFFFF)

Example:

```
/* get boot address */
uint32_t addr = 0;
```

```
addr = efuse_boot_address_get();
```

## efuse\_lock\_config

The description of efuse\_lock\_config is shown as below:

**Table 3-174. Function efuse\_lock\_config**

<b>Function name</b>	efuse_lock_config
<b>Function prototype</b>	void efuse_lock_config(efuse_reg_lock_enum source);
<b>Function descriptions</b>	lock efuse filed
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	specifies filed to lock, refer to <a href="#">Table 3-156. Enum efuse_reg_lock_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock EFUSE boot bits */
efuse_lock_config(EFUSE_BOOT_LOCK);
```

## efuse\_flag\_get

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

**Table 3-175. Function efuse\_flag\_get**

<b>Function name</b>	efuse_flag_get
<b>Function prototype</b>	FlagStatus efuse_flag_get(efuse_flag_enum flag);
<b>Function descriptions</b>	check EFUSE flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specifies to get a flag, refer to <a href="#">Table 3-151. Enum efuse_flag_enum</a>
<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 3-176. Function efuse\_flag\_clear**

<b>Function name</b>	efuse_flag_clear
<b>Function prototype</b>	void efuse_flag_clear(efuse_clear_flag_enum flag);
<b>Function descriptions</b>	clear EFUSE pending flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specifies to clear a flag, refer to <a href="#">Table 3-152. Enum efuse_clear_flag_enum</a>
<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 3-177. Function efuse\_interrupt\_enable**

<b>Function name</b>	efuse_interrupt_enable
<b>Function prototype</b>	void efuse_interrupt_enable(efuse_int_enum source);
<b>Function descriptions</b>	enable EFUSE interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	specifies an interrupt to enable, refer to <a href="#">Table 3-153. Enum efuse_int_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-178. Function efuse\_interrupt\_disable**

<b>Function name</b>	efuse_interrupt_disable
<b>Function prototype</b>	void efuse_interrupt_disable(efuse_int_enum source);
<b>Function descriptions</b>	disable EFUSE interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	specifies an interrupt to disable, refer to <a href="#">Table 3-153. Enum efuse_int_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-179. 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 3-154. Enum efuse_int_flag_enum</a>
<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 3-180. 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_flag);
<b>Function descriptions</b>	clear EFUSE pending interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	specifies to clear a flag, refer to <a href="#">Table 3-155. efuse_clear_int_flag_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear EFUSE write operation complete interrupt flag status */
```

```
efuse_interrupt_flag_clear(EFUSE_INT_PGIC);
```

## 3.9. EXTI

EXTI is the interrupt / event controller in the MCU. It contains up to 26 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.9.1](#), the EXTI firmware functions are introduced in chapter [3.9.2](#).

### 3.9.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

**Table 3-181. EXTI Registers**

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

### 3.9.2. Descriptions of Peripheral functions

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

**Table 3-182. EXTI firmware function**

Function name	Function description
exti_deinit	deinitialize the EXTI
exti_init	initialize EXTI line x
exti_interrupt_enable	enable the interrupts from EXTI line x
exti_interrupt_disable	disable the interrupts from EXTI line x
exti_event_enable	enable the events from EXTI line x
exti_event_disable	disable the events from EXTI line x
exti_software_interrupt_enable	enable the software interrupt event from EXTI line x
exti_software_interrupt_disable	disable the software interrupt event from EXTI line x
exti_flag_get	get EXTI line x interrupt pending flag
exti_flag_clear	clear EXTI line x interrupt pending flag
exti_interrupt_flag_get	get EXTI line x interrupt pending flag
exti_interrupt_flag_clear	clear EXTI line x interrupt pending flag

#### Enum exti\_line\_enum

**Table 3-183. 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_19	EXTI line 19
EXTI_20	EXTI line 20

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

### Enum exti\_mode\_enum

**Table 3-184. Enum exti\_mode\_enum**

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

### Enum exti\_trig\_type\_enum

**Table 3-185. 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	without rising edge or falling edge trigger

### exti\_deinit

The description of exti\_deinit is shown as below:

**Table 3-186. Function exti\_deinit**

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

Example:

```
/* deinitialize the EXTI */
exti_deinit();
```

## exti\_init

The description of exti\_init is shown as below:

**Table 3-187. 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 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 3-183. Enum exti_line_enum</a>
<b>Input parameter{in}</b>	
<b>mode</b>	EXTI mode, refer to <a href="#">Table 3-184. Enum exti_mode_enum</a>
<b>Input parameter{in}</b>	
<b>trig_type</b>	trigger type, refer to <a href="#">Table 3-185. Enum exti_trig_type_enum</a>
<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 3-188. 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 3-183. Enum exti_line_enum</a>
<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 3-189. 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 3-183. Enum exti_line_enum</a>
<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 3-190. 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 3-183. Enum exti_line_enum</a>
<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 3-191. 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 3-183. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the events from EXTI line 0 */
exti_event_disable(EXTI_0);
```

## exti\_software\_interrupt\_enable

The description of exti\_software\_interrupt\_enable is shown as below:

**Table 3-192. 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 software interrupt event 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 3-183. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable EXTI line 0 software interrupt */
exti_software_interrupt_enable(EXTI_0);
```

## exti\_software\_interrupt\_disable

The description of exti\_software\_interrupt\_disable is shown as below:

Table 3-193. Function exti\_software\_interrupt\_disable

Function name	exti_software_interrupt_disable
Function prototype	void exti_software_interrupt_disable(exti_line_enum linex);
Function descriptions	disable the software interrupt event from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-183. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable EXTI line 0 software interrupt */
exti_software_interrupt_disable(EXTI_0);
```

### exti\_flag\_get

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

Table 3-194. Function exti\_flag\_get

Function name	exti_flag_get
Function prototype	FlagStatus exti_flag_get(exti_line_enum linex);
Function descriptions	get EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-183. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

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

### exti\_flag\_clear

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

Table 3-195. Function exti\_flag\_clear

Function name	exti_flag_clear
---------------	-----------------

<b>Function prototype</b>	void exti_flag_clear(exti_line_enum linex);
<b>Function descriptions</b>	clear EXTI line x interrupt 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 3-183. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-196. 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 interrupt 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 3-183. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<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 3-197. 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 interrupt pending flag

Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-183. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EXTI line 0 interrupt flag status */
exti_interrupt_flag_clear(EXTI_0);
```

## 3.10. FMC

There is flash controller and option byte for GD32VW55x series. The FMC registers are listed in chapter [3.10.1](#). The FMC firmware functions are introduced in chapter [3.10.2](#).

### 3.10.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

**Table 3-198. FMC Registers**

Registers	Descriptions
FMC_KEY	FMC unlock key register
FMC_OBKEY	FMC option bytes unlock key register
FMC_STAT	FMC status register
FMC_CTL	FMC control register
FMC_ADDR	FMC address register
FMC_OBSTAT	FMC option bytes status register
FMC_OBR	Option byte register
FMC_OBUSER	Option byte user register
FMC_OBWRP0	Option byte write protection area register 0
FMC_OBWRP1	Option byte write protection area register 1
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_PID0	Product ID0 register
FMC_PID1	Product ID1 register

Registers	Descriptions
FMC_RFT0	RF Trim register 0
FMC_RFT1	RF Trim register 1
FMC_WFTx (x=0..15)	WiFi Trim register x (x=0..15)

### 3.10.2. Descriptions of Peripheral functions

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

**Table 3-199. 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
fmc_obr_function_enable	enable FMC OBR function
fmc_obr_function_disable	disable FMC OBR function
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_user_write	program option bytes USER
ob_write_protection_config	configure write protection pages
fmc_no_rtdec_config	configure NO-RTDEC pages
fmc_offset_region_config	configure offset region
fmc_offset_value_config	configure offset value
fmc_wifi_trim_cal_get	get calibration value
fmc_wifi_trim_pa_get	get Power Amplifier bias tune value
fmc_wifi_trim_get	get wifi trim
fmc_pid_get	get product ID
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
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

Function name	Function description
fmc_interrupt_flag_get	get FMC interrupt flag
fmc_interrupt_flag_clear	clear FMC interrupt flag

## fmc\_state\_enum

Table 3-200. 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_ERR	parameter error

## fmc\_unlock

The description of fmc\_unlock is shown as below:

Table 3-201. 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 3-202. Function fmc\_lock

Function name	fmc_lock
Function prototype	void fmc_lock(void);
Function descriptions	lock the main FMC operation
Precondition	-

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

Example:

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

```
fmc_lock();
```

### fmc\_page\_erase

The description of fmc\_page\_erase is shown as below:

**Table 3-203. Function fmc\_page\_erase**

Function name	fmc_page_erase
Function prototype	fmc_state_enum fmc_page_erase(uint32_t page_address);
Function descriptions	FMC erase page
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
page_address	target page address
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to <a href="#">Table 3-200. 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 3-204. Function fmc\_mass\_erase**

Function name	fmc_mass_erase
Function prototype	fmc_state_enum fmc_mass_erase(void );
Function descriptions	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 3-200. 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 3-205. 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 3-200. 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 3-206. 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 3-200. 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};
```

```
fmc_state_enum state;
```

```
fmc_unlock();
```

```
state = fmc_word_program (0x08004000, data[],16);
```

## fmc\_obr\_function\_enable

The description of fmc\_obr\_function\_enable is shown as below:

**Table 3-207. Function fmc\_obr\_function\_enable**

<b>Function name</b>	fmc_obr_function_enable
<b>Function prototype</b>	void fmc_obr_function_enable(uint32_t obr_function);
<b>Function descriptions</b>	enable FMC OBR function
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>obr_function</b>	the OBR function

<i>FMC_OBR_NWDG_HW</i>	watchdog status function
<i>FMC_OBR_NRST_STDBY</i>	no reset generated when entering Standby mode function
<i>FMC_OBR_NSRT_DPSLP</i>	no reset generated when entering Deepsleep mode function
<i>FMC_OBR_SRAM1_RST</i>	SRAM1 reset automatically function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable software watchdog function */
```

```
ob_unlock();
```

```
fmc_obr_function_enable(FMC_OBR_NWDG_HW);
```

### **fmc\_obr\_function\_disable**

The description of fmc\_obr\_function\_disable is shown as below:

**Table 3-208. Function fmc\_obr\_function\_disable**

<b>Function name</b>	fmc_obr_function_disable
<b>Function prototype</b>	void fmc_obr_function_disable(uint32_t obr_function);
<b>Function descriptions</b>	disable FMC OBR function
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>obr_function</b>	the OBR function
<i>FMC_OBR_NWDG_HW</i>	watchdog status function
<i>FMC_OBR_NRST_STDBY</i>	no reset generated when entering Standby mode function
<i>FMC_OBR_NSRT_DPSLP</i>	no reset generated when entering Deepsleep mode function
<i>FMC_OBR_SRAM1_RST</i>	SRAM1 reset automatically function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable hardware watchdog function */

ob_unlock();

fmc_obr_function_disable(FMC_OBR_NWDG_HW);
```

## ob\_unlock

The description of ob\_unlock is shown as below:

**Table 3-209. 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 3-210. 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 3-211. 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 3-212. 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 3-213. 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_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 3-200. fmc_state_enum</a>

Example:

```
/* configure no security protection */

ob_unlock();

ob_security_protection_config (FMC_NSPC);

ob_start();
```

### ob\_user\_write

The description of ob\_user\_write is shown as below:

**Table 3-214. 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>	
<b>ob_user</b>	option bytes user value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-200. 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 3-215. 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_spage, uint32_t wrp_epage, 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_spage</b>	start page of write protection area, 0~1023
<b>Input parameter{in}</b>	
<b>wrp_epage</b>	end page of write protection area, 0~1023
<b>Input parameter{in}</b>	
<b>wrp_register_index</b>	FMC_OBWRP <sub>x</sub> register index
<b>OBWRP_INDEX0</b>	option byte write protection area register 0
<b>OBWRP_INDEX1</b>	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 3-200. fmc_state_enum</a>

Example:

```
/* configure write protection pages */
```

```
ob_unlock();
```

```
ob_write_protection_config(WRP_REGION_SPAGE, WRP_REGION_EPAGE, OBWRP_IN
DEX0);
```

```
ob_start();
```

### fmc\_no\_rtdec\_config

The description of fmc\_no\_rtdec\_config is shown as below:

**Table 3-216. 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>	
-	-
<b>Return value</b>	
-	-

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 3-217. 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, 0~0x03FF
<b>Input parameter{in}</b>	
<b>of_epage</b>	end page of offset region, 0~0x03FF
<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 3-218. 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>	-
<b>Input parameter{in}</b>	
<b>of_value</b>	offset value, 0~0x1FFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure offset value */
ob_unlock();
fmc_offset_value_config(PAGE_OFFSET_VALUE);
```

## fmc\_wifi\_trim\_cal\_get

The description of fmc\_wifi\_trim\_cal\_get is shown as below:

**Table 3-219. Function fmc\_wifi\_trim\_cal\_get**

<b>Function name</b>	fmc_wifi_trim_cal_get
<b>Function prototype</b>	void fmc_wifi_trim_cal_get(uint32_t *rft0_bletxcal, uint32_t *rft0_wftxcal, uint32_t *rft0_thecal, uint32_t *rft1_wfrxcal);
<b>Function descriptions</b>	get calibration value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
rft0_bletxcal	BLE transmit power calibration value
rft0_wftxcal	WIFI transmit power calibration value
rft0_thecal	thermal meter calibration value
rft1_wfrxcal	WIFI receive gain calibration value
<b>Return value</b>	
-	-

Example:

```
/* get calibration value */

uint32_t bletxcal, wftxcal, thecal, wfrxcal;

fmc_wifi_trim_cal_get(&bletxcal, &wftxcal, &thecal, &wfrxcal);
```

## fmc\_wifi\_trim\_pa\_get

The description of fmc\_wifi\_trim\_pa\_get is shown as below:

**Table 3-220. Function fmc\_wifi\_trim\_pa\_get**

<b>Function name</b>	fmc_wifi_trim_cal_get
<b>Function prototype</b>	void fmc_wifi_trim_pa_get(uint32_t *rft0_pa_tune0, uint32_t *rft0_pa_tune1);
<b>Function descriptions</b>	get Power Amplifier bias tune value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
rft0_pa_tune0	Power Amplifier bias coarse tune value
rft0_pa_tune1	Power Amplifier bias fine tune value
<b>Return value</b>	

Example:

```
/* get Power Amplifier bias tune value */

uint32_t pa_tune0, pa_tune1;

fmc_wifi_trim_pa_get(&pa_tune0, &pa_tune1);
```

### fmc\_wifi\_trim\_get

The description of fmc\_wifi\_trim\_get is shown as below:

**Table 3-221. Function fmc\_wifi\_trim\_get**

<b>Function name</b>	fmc_wifi_trim_get
<b>Function prototype</b>	fmc_state_enum fmc_wifi_trim_get(uint32_t size, uint32_t buf[]);
<b>Function descriptions</b>	configure wifi trim
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>size</b>	the size of data read from WIFI trim register, must be 16U
<b>Output parameter{out}</b>	
<b>buf[ ]</b>	the buffer of data read from WIFI trim register
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Table 3-200. fmc_state_enum</a>

Example:

```
/* get the FMC WIFI trim register */

fmc_state_enum fmc_state = FMC_ERR;

uint32_t size = 16U;

uint32_t data[16];

fmc_state = fmc_wifi_trim_get(size, &data);
```

### ob\_write\_protection\_get

The description of ob\_write\_protection\_get is shown as below:

**Table 3-222. Function ob\_write\_protection\_get**

<b>Function name</b>	ob_write_protection_get
<b>Function prototype</b>	FlagStatus ob_write_protection_get(void);
<b>Function descriptions</b>	get write protection state, only applies to get the status of write/erase protection setting by EFUSE
<b>Precondition</b>	-
<b>The called functions</b>	-

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 3-223. 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
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	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 3-224. Function ob\_security\_protection\_flag\_get**

Function name	ob_security_protection_flag_get
Function prototype	FlagStatus ob_security_protection_flag_get(void);
Function descriptions	get the FMC option bytes security protection state
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check FMC option bytes security protection level 1 is set or not */
```

```
FlagStatus status;
```

```
status = ob_security_protection_flag_get();
```

### fmc\_flag\_get

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

**Table 3-225. Function fmc\_flag\_get**

Function name	fmc_flag_get
Function prototype	FlagStatus fmc_flag_get(uint32_t flag);
Function descriptions	check flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
flag	FMC flag
FMC_FLAG_BUSY	FMC busy flag bit
FMC_FLAG_WPERR	FMC erase/program protection error flag bit
FMC_FLAG_END	FMC end of operation flag bit
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check the FMC_FLAG_END flag set or not*/
```

```
FlagStatus flag;
```

```
flag = fmc_flag_get(FMC_FLAG_END);
```

### fmc\_flag\_clear

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

Table 3-226. Function `fmc_flag_clear`

<b>Function name</b>	<code>fmc_flag_clear</code>
<b>Function prototype</b>	<code>void fmc_flag_clear(uint32_t flag);</code>
<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
<code>FMC_FLAG_WPERR</code>	FMC erase/program protection error flag bit
<code>FMC_FLAG_END</code>	FMC end of operation flag bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-227. Function `fmc_interrupt_enable`

<b>Function name</b>	<code>fmc_interrupt_enable</code>
<b>Function prototype</b>	<code>void fmc_interrupt_enable (uint32_t interrupt);</code>
<b>Function descriptions</b>	enable FMC interrupt
<b>Precondition</b>	<code>fmc_unlock</code>
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the FMC interrupt source
<code>FMC_INT_END</code>	disable FMC end of program interrupt
<code>FMC_INT_ERR</code>	disable FMC error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-228. Function fmc\_interrupt\_disable**

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

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 3-229. 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>	-
<b>Input parameter{in}</b>	
<b>flag</b>	the FMC interrupt flag
<i>FMC_INT_FLAG_WPE</i> <i>RR</i>	FMC erase/program protection error interrupt flag
<i>FMC_INT_FLAG_END</i>	FMC end of operation interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<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 3-230. 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 erase/program protection error interrupt flag
<i>FMC_INT_FLAG_END</i>	FMC end of operation interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear operation interrupt flag */
```

```
fmc_interrupt_flag_clear(FMC_INT_END);
```

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

### 3.11.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

Table 3-231. FWDGT Registers

Registers	Descriptions
FWDGT_CTL	Control register
FWDGT_PSC	Prescaler register
FWDGT_RLD	Reload register
FWDGT_STAT	Status register

### 3.11.2. Descriptions of Peripheral functions

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

Table 3-232. 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_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 3-233. Function fwdgt\_write\_enable

Function name	fwdgt_write_enable
Function prototype	void fwdgt_write_enable(void);
Function descriptions	enable write access to FWDGT_PSC and FWDGT_RLD
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable write access to FWDGT_PSC and FWDGT_RLD */
fwdgt_write_enable();
```

## fwdgt\_write\_disable

The description of fwdgt\_write\_disable is shown as below:

**Table 3-234. 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 3-235. 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 3-236. 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 3-237. 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\_counter\_reload

The description of fwdgt\_counter\_reload is shown as below:

**Table 3-238. 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 3-239. 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:

```
/* confiure 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 3-240. 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);
```

## 3.12. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.12.1](#), the GPIO firmware functions are introduced in chapter [3.12.2](#).

### 3.12.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

**Table 3-241. 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

### 3.12.2. Descriptions of Peripheral functions

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

**Table 3-242. GPIO firmware function**

Function name	Function description
gpio_deinit	reset GPIO port
gpio_mode_set	set GPIO mode
gpio_output_options_set	set GPIO output type and speed
gpio_bit_set	set GPIO pin bit
gpio_bit_reset	reset GPIO pin bit
gpio_bit_write	write data to the specified GPIO pin
gpio_port_write	write data to the specified GPIO port
gpio_input_bit_get	get GPIO pin input status
gpio_input_port_get	get GPIO port input status
gpio_output_bit_get	get GPIO pin output status
gpio_output_port_get	get GPIO port output status
gpio_af_set	set GPIO alternate function
gpio_pin_lock	lock GPIO pin bit
gpio_bit_toggle	toggle GPIO pin status
gpio_port_toggle	toggle GPIO port status

#### gpio\_deinit

The description of gpio\_deinit is shown as below:

Table 3-243. Function gpio\_deinit

Function name	gpio_deinit
Function prototype	void gpio_deinit(uint32_t gpio_periph);
Function descriptions	reset GPIO port
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C)
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 3-244. 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	GPIOx(x = A,B,C)
Input parameter{in}	
mode	gpio pin mode
GPIO_MODE_INPUT	input mode
GPIO_MODE_OUTPUT	output mode
GPIO_MODE_AF	alternate function mode
GPIO_MODE_ANALOG	analog mode
Input parameter{in}	
pull_up_down	gpio pin with pull-up or pull-down resistor
GPIO_PUPD_NONE	floating mode, no pull-up and pull-down resistors

<i>GPIO_PUPD_PULLUP</i>	with pull-up resistor
<i>GPIO_PUPD_PULLDOWN</i>	with pull-down resistor
<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 3-245. 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>	GPIOx(x = A,B,C)
<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
<b>Input parameter{in}</b>	
<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_MAX</i>	output max speed
<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 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 3-246. 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>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C)
<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 */
```

```
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

### gpio\_bit\_reset

The description of gpio\_bit\_reset is shown as below:

Table 3-247. Function gpio\_bit\_reset

<b>Function name</b>	gpio_bit_reset
<b>Function prototype</b>	void gpio_bit_reset(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	reset 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>	GPIOx(x = A,B,C)
<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:

```
/* reset PA0 */
```

```
gpio_bit_reset(GPIOA, GPIO_PIN_0);
```

### gpio\_bit\_write

The description of gpio\_bit\_write is shown as below:

Table 3-248. Function gpio\_bit\_write

<b>Function name</b>	gpio_bit_write
<b>Function prototype</b>	void gpio_bit_write(uint32_t gpio_periph, uint32_t pin, bit_status bit_value);
<b>Function descriptions</b>	write data to the specified GPIO pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C)
<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>Input parameter{in}</b>	
<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 3-249. Function gpio\_port\_write**

Function name	gpio_port_write
Function prototype	void gpio_port_write(uint32_t gpio_periph,uint16_t data);
Function descriptions	write data to the specified GPIO port
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C)
Input parameter{in}	
data	specify the value to be written to the port output data register
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*write 1010 0101 1010 0101 to Port A */
```

```
gpio_port_write(GPIOA, 0xA5A5);
```

## gpio\_input\_bit\_get

The description of gpio\_input\_bit\_get is shown as below:

**Table 3-250. Function gpio\_input\_bit\_get**

Function name	gpio_input_bit_get
Function prototype	FlagStatus gpio_input_bit_get(uint32_t gpio_periph,uint32_t pin);
Function descriptions	get GPIO pin input status
Precondition	-
The called functions	-

Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C)
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	
<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 3-251. 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>	-
Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C)
Output parameter{out}	
-	-
Return value	
<b>uint16_t</b>	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 3-252. 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>	GPIOx(x = A,B,C)
<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>	
<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 3-253. 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>	GPIOx(x = A,B,C)
<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 3-254. 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>	GPIOx(x = A,B,C)
<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>	USART0, UART1, I2C1, CK_OUT0, SPI, JTAG, CK_OUT1, RTC_REF, IFRP_OUT
<i>GPIO_AF_1</i>	TIMER0, TIMER1
<i>GPIO_AF_2</i>	USART0, TIMER0, TIMER2, SPI
<i>GPIO_AF_3</i>	QSPI, USART1, TIMER0, TIMER1, TIMER2
<i>GPIO_AF_4</i>	UART1, SPI, I2C0, QSPI, I2C1
<i>GPIO_AF_5</i>	SPI, I2C0
<i>GPIO_AF_6</i>	TIMER0, I2C0, SPI, I2C1
<i>GPIO_AF_7</i>	USART0, UART1, TIMER16, SPI
<i>GPIO_AF_8</i>	USART0, UART1, UART2, TIMER0, TIMER2, TIMER15
<i>GPIO_AF_9</i>	RTC, TIMER0, TIMER1, TIMER16, IFRP_OUT
<i>GPIO_AF_10</i>	UART2, TIMER2, TIMER16
<i>GPIO_AF_11</i>	TIMER0, TIMER15
<i>GPIO_AF_12</i>	-
<i>GPIO_AF_13</i>	-
<i>GPIO_AF_14</i>	-
<i>GPIO_AF_15</i>	EVENTOUT
<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 3-255. 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>	GPIOx(x = A,B,C)
<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 3-256. 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>	GPIOx(x = A,B,C)

Input parameter{in}	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	GPIO_PIN_ALL
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* toggle PA0 */
```

```
gpio_bit_toggle(GPIOA, GPIO_PIN_0);
```

### gpio\_port\_toggle

The description of gpio\_port\_toggle is shown as below:

**Table 3-257. 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>	-
Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* toggle GPIOA */
```

```
gpio_port_toggle(GPIOA);
```

## 3.13. 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 [3.13.1](#). the HAU firmware functions are introduced in chapter [3.13.2](#).

### 3.13.1. Descriptions of Peripheral registers

HAU registers are listed in the table shown as below:

**Table 3-258. 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

### 3.13.2. Descriptions of Peripheral functions

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

**Table 3-259. 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 structure hau_initpara
hau_reset	reset the HAU processor core
hau_last_word_validbits_num_config	configure the number of valid bits in last word of the message
hau_data_write	write data to the IN FIFO
hau_infifo_words_num_get	return the number of words already written into the IN FIFO
hau_digest_read	read the message digest result
hau_digest_calculation_enable	enable digest calculation
hau_multiple_single_dma_config	configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer or not
hau_dma_enable	enable the HAU DMA interface
hau_dma_disable	disable the HAU DMA interface
hau_context_struct_para_init	initialize the struct context
hau_context_save	save the HAU peripheral context

Function name	Function description
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 3-260. 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 3-261. Structure hau\_digest\_parameter\_struct

Member name	Function description
out[8]	message digest result 0-7

### Structure hau\_context\_parameter\_struct

Table 3-262. 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 3-263. Function hau\_deinit**

<b>Function name</b>	hau_deinit
<b>Function prototype</b>	void hau_deinit(void);
<b>Function descriptions</b>	reset the HAU 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 the HAU peripheral */
```

```
hau_deinit();
```

## hau\_init

The description of hau\_init is shown as below:

**Table 3-264. Function hau\_init**

<b>Function name</b>	hau_init
<b>Function prototype</b>	void hau_init(hau_init_parameter_struct* initpara);
<b>Function descriptions</b>	initialize the HAU peripheral parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
initpara	the HAU peripheral parameters, refer to structure <a href="#">Table 3-260. Structure hau_init_parameter_struct</a>
<b>Return value</b>	
-	-

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 3-265. Function hau\_init\_struct\_para\_init**

<b>Function name</b>	hau_init_struct_para_init
<b>Function prototype</b>	void hau_init_struct_para_init(hau_init_parameter_struct* initpara);
<b>Function descriptions</b>	initialize the struct hau_initpara
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
initpara	the HAU peripheral parameters, refer to structure <a href="#">Table 3-260. Structure hau_init_parameter_struct</a>
<b>Return value</b>	
-	-

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 3-266. Function hau\_reset

Function name	hau_reset
Function prototype	void hau_reset(void);
Function descriptions	reset the HAU processor core
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the HAU processor core */
hau_reset();
```

### hau\_last\_word\_validbits\_num\_config

The description of hau\_last\_word\_validbits\_num\_config is shown as below:

Table 3-267. Function hau\_last\_word\_validbits\_num\_config

Function name	hau_last_word_validbits_num_config
Function prototype	void hau_last_word_validbits_num_config(uint32_t valid_num);
Function descriptions	configure the number of valid bits in last word of the message
Precondition	-
The called functions	-
Input parameter{in}	
valid_num	number of valid bits in last word of the message(0x00 – 0x1F)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the number of valid bits in last word of the message */
hau_last_word_validbits_num_config(0x10);
```

### hau\_data\_write

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

Table 3-268. 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 3-269. 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 3-270. Function hau\_digest\_read

Function name	hau_digest_read
Function prototype	void hau_digest_read(hau_digest_parameter_struct* digestpara);
Function descriptions	read the message digest result
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
digestpara	the message digest result, refer to structure <a href="#">Table 3-261. Structure hau_digest_parameter_struct</a>
Return value	
-	-

Example:

```
/* read the message digest result */
hau_digest_parameter_struct digestpara;
hau_digest_read(&digestpara);
```

### hau\_digest\_calculation\_enable

The description of hau\_digest\_calculation\_enable is shown as below:

Table 3-271. Function hau\_digest\_calculation\_enable

Function name	hau_digest_calculation_enable
Function prototype	void hau_digest_calculation_enable(void);
Function descriptions	enable digest calculation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable digest calculation */
hau_digest_calculation_enable();
```

### hau\_multiple\_single\_dma\_config

The description of hau\_multiple\_single\_dma\_config is shown as below:

Table 3-272. 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 3-273. 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 3-274. 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 3-275. 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 3-262. 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 3-276. 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 3-262. 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 3-277. 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 3-262. Structure hau_context_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-278. 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 */
```

```
static uint8_t output[20];
```

```
ErrStatus status;
```

```
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 3-279. Function hau\_hmac\_sha\_1**

Function name	hau_hmac_sha_1
Function prototype	ErrStatus hau_hmac_sha_1(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[20]);

<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 */

static uint8_t output[20];

ErrStatus status;

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 3-280. 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 */
```

```
static uint8_t output[20];
```

```
ErrStatus status;
```

```
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 3-281. 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 */
```

```
static uint8_t output[20];
```

```
ErrStatus status;
```

```
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 3-282. 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
<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 SHA256 in HASH mode */
```

```
static uint8_t output[20];
```

```
ErrStatus status;
```

```
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 3-283. 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>	-
<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 SHA256 in HMAC mode */

static uint8_t output[20];

ErrStatus status;

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 3-284. 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 */

static uint8_t output[16];

ErrStatus status;

status = hau_hash_md5 (&input, 0x10, output[0]);
```

## hau\_hmac\_md5

The description of hau\_hmac\_md5 is shown as below:

**Table 3-285. 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 */
```

```
static uint8_t output[16];
```

```
ErrStatus status;
```

```
status = hau_hmac_md5 (&key, 0x10, &input, 0x10, output[0]);
```

## hau\_flag\_get

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

**Table 3-286. 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
<i>HAU_FLAG_DATA_INPUT</i>	there is enough space (16 bytes) in the input FIFO
<i>HAU_FLAG_CALCULATION_COMPLETE</i>	digest calculation is completed
<i>HAU_FLAG_DMA</i>	DMA is enabled (DMAE =1) or a transfer is processing
<i>HAU_FLAG_BUSY</i>	data block is in process
<i>HAU_FLAG_INFIFO_NO_EMPTY</i>	the input FIFO is not empty
<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 3-287. 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
HAU_FLAG_DATA_INPUT	there is enough space (16 bytes) in the input FIFO
HAU_FLAG_CALCULATION_COMPLETE	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 3-288. 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>	-

The called functions	-
Input parameter{in}	
<b>interrupt</b>	HAU flag status
<i>HAU_INT_DATA_INPUT_T</i>	a new block can be entered into the IN buffer
<i>HAU_INT_CALCULATION_COMPLETE</i>	calculation complete
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable hau interrupt */
```

```
hau_interrupt_enable(HAU_INT_DATA_INPUT);
```

### **hau\_interrupt\_disable**

The description of `hau_interrupt_disable` is shown as below:

**Table 3-289. Function `hau_interrupt_disable`**

<b>Function name</b>	<code>hau_interrupt_disable</code>
<b>Function prototype</b>	<code>void hau_interrupt_disable(uint32_t interrupt);</code>
<b>Function descriptions</b>	disable the HAU interrupts
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>interrupt</b>	HAU flag status
<i>HAU_INT_DATA_INPUT_T</i>	a new block can be entered into the IN buffer
<i>HAU_INT_CALCULATION_COMPLETE</i>	calculation complete
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable hau interrupt */
```

```
hau_interrupt_disable(HAU_INT_DATA_INPUT);
```

## hau\_interrupt\_flag\_get

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

**Table 3-290. 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 3-291. 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
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>	

Example:

```
/* clear the HAU interrupt flag status */
hau_interrupt_flag_clear(HAU_INT_FLAG_DATA_INPUT);
```

## 3.14. 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 [3.14.1](#), the I2C firmware functions are introduced in chapter [3.14.2](#).

### 3.14.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

**Table 3-292. 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

### 3.14.2. Descriptions of Peripheral functions

I2C firmware functions are listed in the table shown as below:

**Table 3-293. 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

Function name	Function description
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
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_receved_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

Function name	Function description
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
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 3-294. 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 3-295. Function i2c\_deinit**

<b>Function name</b>	i2c_deinit
<b>Function prototype</b>	void i2c_deinit(uint32_t i2c_periph);
<b>Function descriptions</b>	reset I2C
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<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:

```
/* reset I2C0 */
```

```
i2c_deinit(I2C0);
```

## i2c\_timing\_config

The description of i2c\_timing\_config is shown as below:

**Table 3-296. 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>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>psc</b>	0-0xf, timing prescaler
<b>Input parameter{in}</b>	
<b>scl_dely</b>	0-0xf, data setup time
<b>Input parameter{in}</b>	
<b>sda_dely</b>	0-0xf, data hold time
<b>Output parameter{out}</b>	
-	-

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 3-297. Function i2c\_digital\_noise\_filter\_config**

Function name	i2c_digital_noise_filter_config
Function prototype	void i2c_digital_noise_filter_config(uint32_t i2c_periph, uint32_t filter_length);
Function descriptions	configure digital noise filter
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
filter_length	filter_length
FILTER_DISABLE	digital filter is disabled
FILTER_LENGTH_1	digital filter is enabled and filter spikes with a length of up to 1 tI2CCLK
FILTER_LENGTH_2	digital filter is enabled and filter spikes with a length of up to 2 tI2CCLK
FILTER_LENGTH_3	digital filter is enabled and filter spikes with a length of up to 3 tI2CCLK
FILTER_LENGTH_4	digital filter is enabled and filter spikes with a length of up to 4 tI2CCLK
FILTER_LENGTH_5	digital filter is enabled and filter spikes with a length of up to 5 tI2CCLK
FILTER_LENGTH_6	digital filter is enabled and filter spikes with a length of up to 6 tI2CCLK
FILTER_LENGTH_7	digital filter is enabled and filter spikes with a length of up to 7 tI2CCLK
FILTER_LENGTH_8	digital filter is enabled and filter spikes with a length of up to 8 tI2CCLK
FILTER_LENGTH_9	digital filter is enabled and filter spikes with a length of up to 9 tI2CCLK
FILTER_LENGTH_10	digital filter is enabled and filter spikes with a length of up to 10 tI2CCLK
FILTER_LENGTH_11	digital filter is enabled and filter spikes with a length of up to 11 tI2CCLK
FILTER_LENGTH_12	digital filter is enabled and filter spikes with a length of up to 12 tI2CCLK
FILTER_LENGTH_13	digital filter is enabled and filter spikes with a length of up to 13 tI2CCLK
FILTER_LENGTH_14	digital filter is enabled and filter spikes with a length of up to 14 tI2CCLK
FILTER_LENGTH_15	digital filter is enabled and filter spikes with a length of up to 15 tI2CCLK
Output parameter{out}	
-	-
Return value	
-	-

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 3-298. 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 3-299. 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 3-300. 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 3-301. 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 3-302. 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 3-303. 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 3-304. 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 3-305. 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 3-306. 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 3-307. 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 3-308. 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 3-309. 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 3-310. 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 3-311. 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 3-312. 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

Output parameter{out}	
-	-
Return value	
-	-

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 3-313. Function i2c\_address\_bit\_compare\_config**

Function name	i2c_address_bit_compare_config
Function prototype	void i2c_address_bit_compare_config(uint32_t i2c_periph, uint32_t compare_bits);
Function descriptions	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
Precondition	-
The called functions	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	

-	-
---	---

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 3-314. 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 3-315. 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>	

address	I2C address
<b>Input parameter{in}</b>	
<b>addr_mask</b>	the bits not need to compare
<b>ADDRESS2_NO_MASK</b>	no mask, all the bits must be compared
<b>ADDRESS2_MASK_BIT1</b>	ADDRESS2[1] is masked, only ADDRESS2[7:2] are compared
<b>ADDRESS2_MASK_BIT1_2</b>	ADDRESS2[2:1] is masked, only ADDRESS2[7:3] are compared
<b>ADDRESS2_MASK_BIT1_3</b>	ADDRESS2[3:1] is masked, only ADDRESS2[7:4] are compared
<b>ADDRESS2_MASK_BIT1_4</b>	ADDRESS2[4:1] is masked, only ADDRESS2[7:5] are compared
<b>ADDRESS2_MASK_BIT1_5</b>	ADDRESS2[5:1] is masked, only ADDRESS2[7:6] are compared
<b>ADDRESS2_MASK_BIT1_6</b>	ADDRESS2[6:1] is masked, only ADDRESS2[7] are compared
<b>ADDRESS2_MASK_ALL</b>	all the ADDRESS2[7:1] bits are masked
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-316. Function i2c\_second\_address\_disable**

<b>Function name</b>	i2c_second_address_disable
<b>Function prototype</b>	void i2c_second_address_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable i2c second address in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
I2Cx	(x=0,1)
<b>Output parameter{out}</b>	
-	-

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 3-317. Function i2c\_receved\_address\_get**

Function name	i2c_receved_address_get
Function prototype	uint32_t i2c_receved_address_get(uint32_t i2c_periph);
Function descriptions	get received match address in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
uint32_t	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 3-318. Function i2c\_slave\_byte\_control\_enable**

Function name	i2c_slave_byte_control_enable
Function prototype	void i2c_slave_byte_control_enable(uint32_t i2c_periph);
Function descriptions	enable slave byte control
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)

Output parameter{out}	
-	-
Return value	
-	-

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 3-319. Function i2c\_slave\_byte\_control\_disable**

Function name	i2c_slave_byte_control_disable
Function prototype	void i2c_slave_byte_control_disable(uint32_t i2c_periph);
Function descriptions	disable slave byte control
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

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 3-320. Function i2c\_nack\_enable**

Function name	i2c_nack_enable
Function prototype	void i2c_nack_enable(uint32_t i2c_periph);
Function descriptions	generate a NACK in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral

<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

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 3-321. Function i2c\_nack\_disable**

Function name	i2c_nack_disable
Function prototype	void i2c_nack_disable(uint32_t i2c_periph);
Function descriptions	generate a ACK in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

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 3-322. Function i2c\_wakeup\_from\_deepsleep\_enable**

Function name	i2c_wakeup_from_deepsleep_enable
Function prototype	void i2c_wakeup_from_deepsleep_enable(uint32_t i2c_periph);
Function descriptions	enable wakeup from Deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	

<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 3-323. 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 3-324. 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>	-

Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 */
i2c_enable(I2C0);
```

### i2c\_disable

The description of i2c\_disable is shown as below:

**Table 3-325. 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>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 */
i2c_disable(I2C0);
```

### i2c\_start\_on\_bus

The description of i2c\_start\_on\_bus is shown as below:

**Table 3-326. 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>	-

The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 send a start condition to I2C bus */
```

```
i2c_start_on_bus(I2C0);
```

### i2c\_stop\_on\_bus

The description of i2c\_stop\_on\_bus is shown as below:

**Table 3-327. Function i2c\_stop\_on\_bus**

Function name	i2c_stop_on_bus
Function prototype	void i2c_stop_on_bus(uint32_t i2c_periph);
Function descriptions	generate a STOP condition on I2C bus
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 generate a STOP condition to I2C bus */
```

```
i2c_stop_on_bus(I2C0);
```

### i2c\_data\_transmit

The description of i2c\_data\_transmit is shown as below:

**Table 3-328. Function i2c\_data\_transmit**

Function name	i2c_data_transmit
Function prototype	void i2c_data_transmit(uint32_t i2c_periph, uint32_t data);
Function descriptions	I2C transmit data

<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 3-329. 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 3-330. Function i2c\_reload\_enable

Function name	i2c_reload_enable
Function prototype	void i2c_reload_enable(uint32_t i2c_periph);
Function descriptions	enable I2C reload mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C reload mode */
i2c_reload_enable(I2C0);
```

### i2c\_reload\_disable

The description of i2c\_reload\_disable is shown as below:

Table 3-331. Function i2c\_reload\_disable

Function name	i2c_reload_disable
Function prototype	void i2c_reload_disable(uint32_t i2c_periph);
Function descriptions	disable I2C reload mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

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 3-332. Function `i2c_transfer_byte_number_config`

<b>Function name</b>	<code>i2c_transfer_byte_number_config</code>
<b>Function prototype</b>	<code>void i2c_transfer_byte_number_config(uint32_t i2c_periph, uint32_t byte_number);</code>
<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>	
-	-
<b>Return value</b>	
-	-

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 3-333. Function `i2c_dma_enable`

<b>Function name</b>	<code>i2c_dma_enable</code>
<b>Function prototype</b>	<code>void i2c_dma_enable(uint32_t i2c_periph, uint8_t dma);</code>
<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 3-334. 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>	-
<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:

```
/* 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 3-335. 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>	-
<b>Input parameter{in}</b>	
<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 3-336. Function i2c\_pec\_enable**

Function name	i2c_pec_enable
Function prototype	void i2c_pec_enable(uint32_t i2c_periph);
Function descriptions	enable I2C PEC calculation
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C PEC calculation */
```

```
i2c_pec_enable(I2C0);
```

### i2c\_pec\_disable

The description of i2c\_pec\_disable is shown as below:

**Table 3-337. Function i2c\_pec\_disable**

Function name	i2c_pec_disable
Function prototype	void i2c_pec_disable(uint32_t i2c_periph);
Function descriptions	disable I2C PEC calculation
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral

<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C PEC calculation */
```

```
i2c_pec_disable(I2C0);
```

### i2c\_pec\_value\_get

The description of i2c\_pec\_value\_get is shown as below:

**Table 3-338. Function i2c\_pec\_value\_get**

Function name	i2c_pec_value_get
Function prototype	uint32_t i2c_pec_value_get(uint32_t i2c_periph);
Function descriptions	get packet error checking value
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
uint32_t	PEC value

Example:

```
/* I2C0 get packet error checking value */
```

```
uint32_t pec_value;
```

```
pec_value = i2c_pec_value_get(I2C0);
```

### i2c\_smbus\_alert\_enable

The description of i2c\_smbus\_alert\_enable is shown as below:

**Table 3-339. Function i2c\_smbus\_alert\_enable**

Function name	i2c_smbus_alert_enable
Function prototype	void i2c_smbus_alert_enable(uint32_t i2c_periph);
Function descriptions	enable SMBus Alert
Precondition	-
The called functions	-

Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

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 3-340. Function i2c\_smbus\_alert\_disable**

<b>Function name</b>	i2c_smbus_alert_disable
<b>Function prototype</b>	void i2c_smbus_alert_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable SMBus Alert
<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:

```
/* 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 3-341. Function i2c\_smbus\_default\_addr\_enable**

<b>Function name</b>	i2c_smbus_default_addr_enable
<b>Function prototype</b>	void i2c_smbus_default_addr_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable SMBus device default address
<b>Precondition</b>	-

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 3-342. 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 3-343. 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

<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 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 3-344. Function i2c\_smbus\_host\_addr\_disable**

<b>Function name</b>	i2c_smbus_host_addr_disable
<b>Function prototype</b>	void i2c_smbus_host_addr_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable SMBus Host 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>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-345. Function i2c\_extented\_clock\_timeout\_enable**

<b>Function name</b>	i2c_extented_clock_timeout_enable
<b>Function prototype</b>	void i2c_extented_clock_timeout_enable(uint32_t i2c_periph);

<b>Function descriptions</b>	enable extended 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:

```
/* 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 3-346. Function i2c\_extented\_clock\_timeout\_disable**

<b>Function name</b>	i2c_extented_clock_timeout_disable
<b>Function prototype</b>	void i2c_extented_clock_timeout_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable extended 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 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 3-347. Function i2c\_clock\_timeout\_enable**

<b>Function name</b>	i2c_clock_timeout_enable
----------------------	--------------------------

<b>Function prototype</b>	void i2c_clock_timeout_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable 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:

```
/* 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 3-348. 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 3-349. 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 3-350. 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>	0-0xffff, 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 3-351. 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 3-352. 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
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>flag</b>	I2C flags
<i>I2C_FLAG_TBE</i>	I2C_TDATA is empty during transmitting

<i>I2C_FLAG_TI</i>	transmit interrupt
<i>I2C_FLAG_RBNE</i>	I2C_RDATA is not empty during receiving
<i>I2C_FLAG_ADDSEND</i>	address received matches in slave mode
<i>I2C_FLAG_NACK</i>	not acknowledge flag
<i>I2C_FLAG_STPDET</i>	STOP condition detected in slave mode
<i>I2C_FLAG_TC</i>	transfer complete in master mode
<i>I2C_FLAG_TCR</i>	transfer complete reload
<i>I2C_FLAG_BERR</i>	bus error
<i>I2C_FLAG_LOSTARB</i>	arbitration Lost
<i>I2C_FLAG_OUERR</i>	overflow/underrun error in slave mode
<i>I2C_FLAG_PECERR</i>	PEC error
<i>I2C_FLAG_TIMEOUT</i>	timeout flag
<i>I2C_FLAG_SMBALT</i>	SMBus Alert
<i>I2C_FLAG_I2CBSY</i>	busy flag
<i>I2C_FLAG_TR</i>	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 3-353. 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
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>flag</b>	I2C flags
<i>I2C_FLAG_ADDSEND</i>	address received matches in slave mode
<i>I2C_FLAG_NACK</i>	not acknowledge flag

<i>I2C_FLAG_STPDET</i>	STOP condition detected in slave mode
<i>I2C_FLAG_BERR</i>	bus error
<i>I2C_FLAG_LOSTARB</i>	arbitration Lost
<i>I2C_FLAG_OUERR</i>	overrun/underrun error in slave mode
<i>I2C_FLAG_PECERR</i>	PEC error
<i>I2C_FLAG_TIMEOUT</i>	timeout flag
<i>I2C_FLAG_SMBALT</i>	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 3-354. 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 3-355. 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 3-356. 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 3-294. 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 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 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 RB</i>	arbitration lost interrupt flag
<i>I2C_INT_FLAG_OUER R</i>	overflow/underrun error in slave mode interrupt flag
<i>I2C_INT_FLAG_PECE RR</i>	PEC error interrupt flag
<i>I2C_INT_FLAG_TIMEO UT</i>	timeout interrupt flag
<i>I2C_INT_FLAG_SMBA 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 3-357. Function `i2c_interrupt_flag_clear`

<b>Function name</b>	<code>i2c_interrupt_flag_clear</code>
<b>Function prototype</b>	<code>void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);</code>
<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 3-294. i2c_interrupt_flag_enum</a> .
<i>I2C_INT_FLAG_ADDS 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 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 RB</i>	arbitration lost interrupt flag
<i>I2C_INT_FLAG_OUER R</i>	overflow/underrun error in slave mode interrupt flag
<i>I2C_INT_FLAG_PECE RR</i>	PEC error interrupt flag
<i>I2C_INT_FLAG_TIMEO UT</i>	timeout interrupt flag
<i>I2C_INT_FLAG_SMBA LT</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);
```

## 3.15. 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 [3.15.1](#),

the PKCAU firmware functions are introduced in chapter [3.15.2](#).

### 3.15.1. Descriptions of Peripheral registers

PKCAU registers are listed in the table shown as below:

**Table 3-358. PKCAU Registers**

Registers	Descriptions
PKCAU_CTL	Control register
PKCAU_STAT	Status register
PKCAU_STATC	Status clear register

### 3.15.2. Descriptions of Peripheral functions

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

**Table 3-359. 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_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_ecc_out_struct_para_init	initialize ecc output 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

Function name	Function description
pkcau_ecdsa_sign_operation	execute ECDSA sign operation
pkcau_ecdsa_verification_operation	execute ECDSA verify operation
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
pkcau_interrupt_flag_clear	clear PKCAU interrupt flag status

### Structure pkcau\_mont\_parameter\_struct

Table 3-360. Structure pkcau\_mont\_parameter\_struct

Member name	Function description
modulus_n	modulus value n
modulus_n_len	modulus length in byte

### Structure pkcau\_mod\_parameter\_struct

Table 3-361. Structure pkcau\_mod\_parameter\_struct

Member name	Function description
opr_d_a	operand A
opr_d_a_len	operand A length in byte
opr_d_b	operand B
opr_d_b_len	operand b length in byte
modulus_n	modulus value n
modulus_n_len	modulus length in byte

### Structure pkcau\_mod\_exp\_parameter\_struct

Table 3-362. Structure pkcau\_mod\_exp\_parameter\_struct

Member name	Function description
opr_d_a	operand A
opr_d_a_len	operand A length in byte
exp_e	exponent e
e_len	exponent length in byte
modulus_n	modulus n
modulus_n_len	modulus length in byte
mont_para	montgomery parameter R2 mod n
mont_para_len	montgomery parameter length in byte

### Structure pkcau\_arithmetic\_parameter\_struct

**Table 3-363. Structure pkcau\_arithmetic\_parameter\_struct**

Member name	Function description
opr_d_a	operand A
opr_d_a_len	length of operand A in byte
opr_d_b	operand B
opr_d_b_len	length of operand B in byte

### Structure pkcau\_crt\_parameter\_struct

**Table 3-364. Structure pkcau\_crt\_parameter\_struct**

Member name	Function description
opr_d_a	operand A
opr_d_a_len	length of operand A in byte
opr_d_dp	operand dp
opr_d_dp_len	length of operand dp in byte
opr_d_dq	operand dq
opr_d_dq_len	length of operand dq in byte
opr_d_qinv	operand qinv
opr_d_qinv_len	length of operand qinv in byte
opr_d_p	prime operand p
opr_d_p_len	length of operand p in byte
opr_d_q	prime operand q
opr_d_q_len	length of operand q in byte

### Structure pkcau\_ec\_group\_parameter\_struct

**Table 3-365. Structure pkcau\_ec\_group\_parameter\_struct**

Member name	Function description
modulus_p	curve modulus p
modulus_p_len	curve modulus p length in byte
coff_a	curve coefficient a
coff_a_len	curve coefficient a length in byte
coff_b	curve coefficient b
coff_b_len	curve coefficient b length in byte
base_point_x	curve base point coordinate x
base_point_x_len	curve base point coordinate x length in byte
base_point_y	curve base point coordinate y
base_point_y_len	curve base point coordinate y length in byte
order_n	curve prime order n
order_n_len	curve prime order n length in byte
a_sign	curve coefficient a sign

Member name	Function description
multi_k	scalar multiplier k
multi_k_len	length of scalar multiplier k
integer_k	integer k
integer_k_len	integer k length in byte
private_key_d	private key d
private_key_d_len	private key d length in byte
mont_para	montgomery parameter R2 mod n
mont_para_len	montgomery parameter R2 mod n length in byte

### Structure pkcau\_point\_parameter\_struct

Table 3-366. Structure pkcau\_point\_parameter\_struct

Member name	Function description
point_x	point coordinate x
point_x_len	point coordinate x length in byte
point_y	point coordinate y
point_y_len	point coordinate y length in byte

### Structure pkcau\_signature\_parameter\_struct

Table 3-367. Structure pkcau\_signature\_parameter\_struct

Member name	Function description
sign_r	signature part r
sign_r_len	signature part r length in byte
sign_s	signature part s
sign_s_len	signature part s length in byte

### Structure pkcau\_hash\_parameter\_struct

Table 3-368. Structure pkcau\_hash\_parameter\_struct

Member name	Function description
hash_z	hash value z
hash_z_len	hash value z length in byte

### Structure pkcau\_ecc\_out\_struct

Table 3-369. Structure pkcau\_ecc\_out\_struct

Member name	Function description
sign_extra	flag of extended ECDSA sign (extra outputs)
sign_r	signature part r
sign_s	signature part s
point_x	point coordinate x
point_y	point coordinate y

## pkcau\_deinit

The description of pkcau\_deinit is shown as below:

**Table 3-370. Function pkcau\_deinit**

<b>Function name</b>	pkcau_deinit
<b>Function prototype</b>	void pkcau_deinit(void);
<b>Function descriptions</b>	reset PKCAU
<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 PKCAU */
pkcau_deinit();
```

## pkcau\_mont\_struct\_para\_init

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

**Table 3-371. Function pkcau\_mont\_struct\_para\_init**

<b>Function name</b>	pkcau_mont_struct_para_init
<b>Function prototype</b>	void pkcau_mont_struct_para_init(pkcau_mont_parameter_struct* init_para);
<b>Function descriptions</b>	initialize montgomery 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>	montgomery parameter struct, the structure members can refer to <a href="#">Table 3-360. Structure pkcau_mont_parameter_struct</a> .
<b>Return value</b>	
-	-

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 3-372. Function pkcau\_mod\_struct\_para\_init**

<b>Function name</b>	pkcau_mod_struct_para_init
<b>Function prototype</b>	void pkcau_mod_struct_para_init(pkcau_mod_parameter_struct* init_para);
<b>Function descriptions</b>	initialize modular 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 parameter struct, the structure members can refer to <a href="#">Table 3-361. Structure pkcau_mod_parameter_struct.</a>
<b>Return value</b>	
-	-

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 3-373. 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 3-362. Structure pkcau_mod_exp_parameter_struct.</a>
<b>Return value</b>	
-	-

Example:

```
/* initialize PKCAU modular exponentiation 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\_arithmetic\_struct\_para\_init

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

**Table 3-374. 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>	
<b>init_para</b>	arithmetic parameter struct, the structure members can refer to <a href="#">Table 3-363. 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 3-375. 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 3-364. 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 3-376. 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 3-365. 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 3-377. 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

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	point parameter struct, the structure members can refer to <a href="#">Table 3-366.</a> <a href="#">Structure pkcau_point_parameter_struct.</a>
Return value	
-	-

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 3-378. Function pkcau\_signature\_struct\_para\_init**

Function name	pkcau_signature_struct_para_init
Function prototype	void pkcau_signature_struct_para_init(pkcau_signature_parameter_struct* init_para);
Function descriptions	initialize signature parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	signature parameter struct, the structure members can refer to <a href="#">Table 3-367.</a> <a href="#">Structure pkcau_signature_parameter_struct.</a>
Return value	
-	-

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 3-379. 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 3-368. 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\_ecc\_out\_struct\_para\_init

The description of pkcau\_ecc\_out\_struct\_para\_init is shown as below:

Table 3-380. Function pkcau\_ecc\_out\_struct\_para\_init

Function name	pkcau_ecc_out_struct_para_init
Function prototype	void pkcau_ecc_out_struct_para_init(pkcau_ecc_out_struct* init_para)
Function descriptions	initialize ECC out parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	ecdsa signature, ecc scalar multiplication output structure, the structure members can refer to <a href="#">Table 3-369. Structure pkcau_ecc_out_struct</a> .
Return value	
-	-

Example:

```
/* initialize PKCAU modular reduction parameter struct with a default value */
```

```
pkcau_ecc_out_struct pkcau_ecc_out_parameter;
```

```
pkcau_ecc_out_struct_para_init(&pkcau_ecc_out_parameter);
```

## pkcau\_enable

The description of pkcau\_enable is shown as below:

**Table 3-381. 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 3-382. 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 3-383. 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 3-384. 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_INVERSION</i>	modular inversion
<i>PKCAU_MODE_ARITH</i>	arithmetic addition

<i>METIC_ADD</i>	
<i>PKCAU_MODE_ARITH METIC_SUB</i>	arithmetic subtraction
<i>PKCAU_MODE_ARITH METIC_MUL</i>	arithmetic multiplication
<i>PKCAU_MODE_ARITH METIC_COMP</i>	arithmetic comparison
<i>PKCAU_MODE_MOD_ REDUCTION</i>	modular reduction
<i>PKCAU_MODE_MOD_ ADD</i>	modular addition
<i>PKCAU_MODE_MOD_ SUB</i>	modular subtraction
<i>PKCAU_MODE_MONT _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_ECDS A_SIGN</i>	ECDSA sign
<i>PKCAU_MODE_ECDS 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 3-385. 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, uint8_t* results)
<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 3-360. Structure pkcau_mont_parameter_struct.</a>
<b>Output parameter{out}</b>	
<b>results</b>	output buffer
<b>Return value</b>	
-	-

Example:

```
uint8_t results[ME_MOD_SIZE];

/* 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_n_len = ME_MOD_SIZE;
pkcau_mont_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute montgomery parameter operation */

pkcau_mont_param_operation(&pkcau_mont_parameter, results);
```

### pkcau\_mod\_operation

The description of pkcau\_mod\_operation is shown as below:

**Table 3-386. Function pkcau\_mod\_operation**

<b>Function name</b>	pkcau_mod_operation
<b>Function prototype</b>	pkcau_mod_operation(pkcau_mod_parameter_struct* mod_para, uint32_t mode, uint8_t* results)
<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 3-361. Structure pkcau_mod_parameter_struct.</a>
<b>Input parameter{in}</b>	
<b>mode</b>	modular operation mode
<b>PKCAU_MODE_MOD_ADD</b>	modular addition

<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>results</b>	output buffer
<b>Return value</b>	
-	-

Example:

```
uint8_t results[MA_MOD_SIZE];

/* 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_n_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, results);
```

### pkcau\_mod\_exp\_operation

The description of pkcau\_mod\_exp\_operation is shown as below:

**Table 3-387. Function pkcau\_mod\_exp\_operation**

<b>Function name</b>	pkcau_mod_exp_operation
<b>Function prototype</b>	pkcau_mod_exp_operation(pkcau_mod_exp_parameter_struct* mod_exp_para, uint32_t mode, uint8_t* results)
<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 3-362. 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>results</b>	output buffer
<b>Return value</b>	
-	-

Example:

```
uint8_t mod_exp_res[ME_MOD_SIZE];

/* 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.oprd_a_len = sizeof(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_n_len = ME_MOD_SIZE;
pkcau_mod_exp_parameter.modulus_n = (uint8_t *)modulus_n;
pkcau_mod_exp_parameter.mont_para = (uint8_t *)mont_para;
pkcau_mod_exp_parameter.mont_para_len = sizeof(mont_para);

/* execute modular exponentiation fast operation */

pkcau_mod_exp_operation(&pkcau_mod_exp_parameter,
PKCAU_MODE_MOD_EXP_FAST, mod_exp_res);
```

### pkcau\_mod\_inver\_operation

The description of pkcau\_mod\_inver\_operation is shown as below:

**Table 3-388. Function pkcau\_mod\_inver\_operation**

<b>Function name</b>	pkcau_mod_inver_operation
<b>Function prototype</b>	pkcau_mod_inver_operation(pkcau_mod_parameter_struct* mod_inver_para, uint8_t* results)
<b>Function descriptions</b>	execute modular inversion operation
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>mod_inver_para</b>	modular inversion parameter struct, the structure members can refer to <a href="#">Table 3-361. Structure pkcau_mod_parameter_struct.</a>
Output parameter{out}	
<b>results</b>	output buffer
Return value	
-	-

Example:

```
uint8_t mod_inver_res[ME_MOD_SIZE];

/* initialize the modular inversion parameter structure */

pkcau_mod_parameter_struct pkcau_mod_inver_parameter;

pkcau_mod_inver_struct_para_init(&pkcau_mod_inver_parameter);

/* initialize the modular parameters */

pkcau_mod_inver_parameter.oprd_a = (uint8_t *)oprd_a;

pkcau_mod_inver_parameter.oprd_a = sizeof(oprd_a);

pkcau_mod_inver_parameter.modulus_n_len = ME_MOD_SIZE;

pkcau_mod_inver_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute modular inversion operation */

pkcau_mod_inver_operation(&pkcau_mod_inver_parameter, mod_inver_res);
```

### pkcau\_mod\_reduc\_operation

The description of pkcau\_mod\_reduc\_operation is shown as below:

**Table 3-389. Function pkcau\_mod\_reduc\_operation**

<b>Function name</b>	pkcau_mod_reduc_operation
<b>Function prototype</b>	void pkcau_mod_reduc_operation(pkcau_mod_parameter_struct* mod_reduc_para, uint8_t* results)
<b>Function descriptions</b>	execute modular reduction operation
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>mod_reduc_para</b>	modular reduction parameter struct, the structure members can refer to <a href="#">Table 3-361. Structure pkcau_mod_parameter_struct.</a>
Output parameter{out}	
<b>results</b>	output buffer
Return value	
-	-

Example:

```
uint8_t mod_reduc_res[MA_MOD_SIZE];

/* initialize the modular inversion parameter structure */

pkcau_mod_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_n_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, mod_reduc_res);
```

### pkcau\_arithmetic\_operation

The description of pkcau\_arithmetic\_operation is shown as below:

**Table 3-390. 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, uint8_t* results)
<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 3-363. 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>	

results	output buffer
Return value	
-	-

Example:

```
uint8_t ari_multi_res[AM_SIZE];

/* 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_a_len = AM_A_SIZE;

pkcau_ari_multi_parameter.oprd_b = (uint8_t *)oprd_b;

pkcau_ari_multi_parameter.oprd_b_len = AM_B_SIZE;

/* execute arithmetic addition operation */

pkcau_arithmetic_operation(&pkcau_arithmetic_parameter,
PKCAU_MODE_ARITHMETIC_ADD, ari_multi_res);
```

### pkcau\_crt\_exp\_operation

The description of pkcau\_crt\_exp\_operation is shown as below:

**Table 3-391. Function pkcau\_crt\_exp\_operation**

Function name	pkcau_crt_exp_operation
Function prototype	void pkcau_crt_exp_operation(pkcau_crt_parameter_struct* crt_para, uint8_t* results);
Function descriptions	execute RSA CRT exponentiation operation
Precondition	-
The called functions	-
Input parameter{in}	
crt_para	CRT parameter struct, the structure members can refer to <a href="#">Table 3-364. Structure pkcau crt parameter struct.</a>
Output parameter{out}	
results	output buffer
Return value	
-	-

Example:

```
uint8_t crt_res[sizeof(rsa_crt_a)];
```

```

/* 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_a_len = sizeof(rsa_crt_a);
pkcau_crt_parameter.oprd_dp = (uint8_t *)rsa_crt_dp;
pkcau_crt_parameter.oprd_dp_len = sizeof(rsa_crt_dp);
pkcau_crt_parameter.oprd_dq = (uint8_t *)rsa_crt_dq;
pkcau_crt_parameter.oprd_dq_len = sizeof(rsa_crt_dq);
pkcau_crt_parameter.oprd_qinv = (uint8_t *)rsa_crt_qinv;
pkcau_crt_parameter.oprd_qinv_len = sizeof(rsa_crt_qinv);
pkcau_crt_parameter.oprd_p = (uint8_t *)rsa_crt_p;
pkcau_crt_parameter.oprd_p_len = sizeof(rsa_crt_p);
pkcau_crt_parameter.oprd_q = (uint8_t *)rsa_crt_q;
pkcau_crt_parameter.oprd_q_len = sizeof(rsa_crt_q);

/* execute RSA CRT exponentiation operation */

pkcau_crt_exp_operation(&pkcau_crt_parameter, crt_res);

```

### pkcau\_point\_check\_operation

The description of pkcau\_point\_check\_operation is shown as below:

**Table 3-392. Function pkcau\_point\_check\_operation**

<b>Function name</b>	pkcau_point_check_operation
<b>Function prototype</b>	uint8_t 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 3-366. 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 3-365. Structure pkcau_ec_group_parameter_struct</a> .

Output parameter{out}	
-	-
Return value	
uint8_t	flag indicating whether the point is on an elliptic curve or not

Example:

```
uint8_t res = 1;

/* 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_x_len = sizeof(pcheck_x);
pkcau_point_check_parameter.point_y = pcheck_y;
pkcau_point_check_parameter.point_y_len = sizeof(pcheck_y);

/* initialize the input ECC curve parameter */
pkcau_curve_group.modulus_p = (uint8_t *)brainpoolp256r1_p;
pkcau_curve_group.modulus_p_len = sizeof(brainpoolp256r1_p);
pkcau_curve_group.coff_a = (uint8_t *)brainpoolp256r1_a;
pkcau_curve_group.coff_a_len = sizeof(brainpoolp256r1_a);
pkcau_curve_group.coff_b = (uint8_t *)brainpoolp256r1_b;
pkcau_curve_group.coff_b_len = sizeof(brainpoolp256r1_b);
pkcau_curve_group.a_sign = 0;

/* execute point check operation */
res = 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 3-393. Function pkcau\_point\_mul\_operation**

Function name	pkcau_point_mul_operation
---------------	---------------------------

<b>Function prototype</b>	void pkcau_point_mul_operation(pkcau_point_parameter_struct* point_para, const pkcau_ec_group_parameter_struct* curve_group_para, \n uint32_t mode, pkcau_ecc_out_struct* result);
<b>Function descriptions</b>	execute point multiplication 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 3-366. 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 3-365. Structure pkcau_ec_group_parameter_struct.</a>
<b>Input parameter{in}</b>	
<b>mode</b>	point multiplication operation mode
<i>PKCAU_MODE_ECC_MUL</i>	montgomery parameter computation then ECC scalar multiplication
<i>PKCAU_MODE_ECC_MUL_FAST</i>	ECC scalar multiplication only
<b>Output parameter{out}</b>	
<b>result</b>	ecdsa signature, ecc scalar multiplication output structure, the structure members can refer to <a href="#">Table 3-369. Structure pkcau_ecc_out_struct.</a>
<b>Return value</b>	
-	-

Example:

```
pkcau_ecc_out_struct pkcau_ecc_out_result;

/* initialize the ECC out parameter */

pkcau_ecc_out_struct_para_init(&pkcau_ecc_out_result);

pkcau_ecc_out_result.point_x = res_x;

pkcau_ecc_out_result.point_y = res_y;

/* 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_x_len = sizeof(ec_pmul_x);

pkcau_point_parameter.point_y = ec_pmul_y;

pkcau_point_parameter.point_x_len = sizeof(ec_pmul_x);

/* initialize the input ECC curve parameter */

pkcau_curve_group.modulus_p      = (uint8_t *)brainpoolp256r1_p;
pkcau_curve_group.modulus_p_len = sizeof(brainpoolp256r1_p);
pkcau_curve_group.coff_a         = (uint8_t *)brainpoolp256r1_a;
pkcau_curve_group.coff_a_len     = sizeof(brainpoolp256r1_a);
pkcau_curve_group.a_sign         = 0;
pkcau_curve_group.multi_k        = ec_pmul_k;
pkcau_curve_group.multi_k_len    = PMUL_K_SIZE;

/* execute scalar multiplication operation */

pkcau_point_mul_operation(&pkcau_point_parameter, &pkcau_curve_group,
PKCAU_MODE_ECC_MUL, pkcau_ecc_out_result);

```

## pkcau\_ecdsa\_sign\_operation

The description of pkcau\_ecdsa\_sign\_operation is shown as below:

**Table 3-394. Function pkcau\_ecdsa\_sign\_operation**

Function name	pkcau_ecdsa_sign_operation
Function prototype	uint8_t pkcau_ecdsa_sign_operation(pkcau_hash_parameter_struct* hash_para, \ const pkcau_ec_group_parameter_struct* curve_group_para, \ pkcau_ecc_out_struct* result);
Function descriptions	execute ECDSA sign operation
Precondition	-
The called functions	-
Input parameter{in}	
hash_para	hash parameter struct, the structure members can refer to <a href="#">Table 3-368. 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 3-365. Structure pkcau_ec_group_parameter_struct</a> .
Output parameter{out}	
result	ecdsa signature, ecc scalar multiplication output structure, the structure members can refer to <a href="#">Table 3-369. Structure pkcau_ecc_out_struct</a> .
Return value	

<b>uint8_t</b>	flag indicating whether the signature operation was successful
----------------	--

Example:

```
pkcau_ecc_out_struct pkcau_ecc_out_result;

/* initialize the ECC out parameter */

pkcau_ecc_out_struct_para_init(&pkcau_ecc_out_result);

pkcau_ecc_out_result.sign_r = r;

pkcau_ecc_out_result.sign_s = s;

/* 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.modulus_p_len = sizeof(secp112r2_p);

pkcau_curve_group.coff_a         = secp112r2_a;

pkcau_curve_group.coff_a_len     = sizeof(secp112r2_a);

pkcau_curve_group.a_sign         = 0;

pkcau_curve_group.base_point_x   = secp112r2_gx;

pkcau_curve_group.base_point_x_len = sizeof(secp112r2_gx);

pkcau_curve_group.base_point_y   = secp112r2_gy;

pkcau_curve_group.base_point_y_len = sizeof(secp112r2_gy);

pkcau_curve_group.order_n        = secp112r2_n,

pkcau_curve_group.order_n_len    = sizeof(secp112r2_n);

pkcau_curve_group.integer_k      = k;

pkcau_curve_group.integer_k_len  = DATA_SIZE;
```

```
pkcau_curve_group.private_key_d      = d;

pkcau_curve_group.private_key_d_len = DATA_SIZE;

/* execute ECDSA sign operation */

pkcau_ecdsa_sign_operation(&pkcau_hash_parameter, &pkcau_curve_group, results);
```

## pkcau\_ecdsa\_verification\_operation

The description of pkcau\_ecdsa\_verification\_operation is shown as below:

**Table 3-395. Function pkcau\_ecdsa\_verification\_operation**

Function name	pkcau_ecdsa_verification_operation
Function prototype	uint8_t 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)
Function descriptions	execute ECDSA verification operation
Precondition	-
The called functions	-
Input parameter{in}	
point_para	point parameter struct, the structure members can refer to <a href="#">Table 3-366. Structure pkcau_point_parameter_struct.</a>
Input parameter{in}	
hash_para	hash parameter struct, the structure members can refer to <a href="#">Table 3-368. Structure pkcau_hash_parameter_struct.</a>
Input parameter{in}	
signature_para	signature parameter struct, the structure members can refer to <a href="#">Table 3-367. Structure pkcau_signature_parameter_struct.</a>
Input parameter{in}	
curve_group_para	ECC curve parameter struct, the structure members can refer to <a href="#">Table 3-365. Structure pkcau_ec_group_parameter_struct.</a>
Output parameter{out}	
-	-
Return value	
uint8_t	flag indicating whether the signature verification operation was successful

Example:

```
uint8_t verify_res = 1;

/* ECC curve parameter structure */

pkcau_ec_group_parameter_struct pkcau_curve_group;

/* hash parameter structure */
```

```
pkcau_hash_parameter_struct pkcau_hash_parameter;

/* signature parameter structure */

pkcau_signature_parameter_struct pkcau_signature_parameter;

/* point parameter structure */

pkcau_point_parameter_struct pkcau_point_parameter;

/* initialize the ECC curve parameter, hash parameter, point parameter and signature
parameter structure */

pkcau_ec_group_struct_para_init(&pkcau_curve_group);

pkcau_hash_struct_para_init(&pkcau_hash_parameter);

pkcau_point_struct_para_init(&pkcau_point_parameter);

pkcau_signature_struct_para_init(&pkcau_signature_parameter);

/* initialize the input ECC signature parameters */

pkcau_signature_parameter.sign_r      = (uint8_t *)ecc_verify_r;
pkcau_signature_parameter.sign_r_len = sizeof(ecc_verify_r);
pkcau_signature_parameter.sign_s      = (uint8_t *)ecc_verify_s;
pkcau_signature_parameter.sign_s_len = sizeof(ecc_verify_s);

/* initialize the input point parameters */

pkcau_point_parameter.point_x         = ecc_verify_x;
pkcau_point_parameter.point_x_len    = sizeof(ecc_verify_x);
pkcau_point_parameter.point_y         = ecc_verify_y;
pkcau_point_parameter.point_y_len    = sizeof(ecc_verify_y);

/* initialize the input ECC curve parameters */

pkcau_curve_group.modulus_p           = (uint8_t *)brainpoolp256r1_p;
pkcau_curve_group.modulus_p_len      = sizeof(brainpoolp256r1_p);
pkcau_curve_group.coff_a              = (uint8_t *)brainpoolp256r1_a;
pkcau_curve_group.coff_a_len         = sizeof(brainpoolp256r1_a);
pkcau_curve_group.a_sign              = 0;
pkcau_curve_group.base_point_x        = (uint8_t *)brainpoolp256r1_gx;
pkcau_curve_group.base_point_x_len    = sizeof(brainpoolp256r1_gx);
pkcau_curve_group.base_point_y        = (uint8_t *)brainpoolp256r1_gy;
```

```

pkcau_curve_group.base_point_y_len = sizeof(brainpoolp256r1_gy);

pkcau_curve_group.order_n          = (uint8_t *)brainpoolp256r1_n,

pkcau_curve_group.order_n_len     = sizeof(brainpoolp256r1_n);

/* 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);

/* execute ECDSA verification operation */

verify_res = pkcau_ecdsa_verification_operation(&pkcau_point_parameter,
&pkcau_hash_parameter, &pkcau_signature_parameter, &pkcau_curve_group);

```

### pkcau\_flag\_get

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

**Table 3-396. Function pkcau\_flag\_get**

Function name	pkcau_flag_get
Function prototype	FlagStatus pkcau_flag_get(uint32_t flag);
Function descriptions	get PKCAU flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	PKCAU flags
PKCAU_FLAG_ADDRERR	address error flag
PKCAU_FLAG_RAMERR	PKCAU RAM error flag
PKCAU_FLAG_END	end of PKCAU operation flag
PKCAU_FLAG_BUSY	busy flag
Output parameter{out}	
-	-
Return value	
FlagStatus	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 3-397. 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_ADDRERR</i>	address error flag
<i>PKCAU_FLAG_RAMERR</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 3-398. 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_ADDRERR</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>	
-	-

Return value	
-	-

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 3-399. Function pkcau\_interrupt\_disable**

Function name	pkcau_interrupt_disable
Function prototype	void pkcau_interrupt_disable(uint32_t interrupt);
Function descriptions	disable PKCAU interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	interrupt type
<i>PKCAU_INT_ADDRERR</i>	address error interrupt
<i>PKCAU_INT_RAMERR</i>	PKCAU RAM error interrupt
<i>PKCAU_INT_END</i>	end of PKCAU operation interrupt
Output parameter{out}	
-	-
Return value	
-	-

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 3-400. Function pkcau\_interrupt\_flag\_get**

Function name	pkcau_interrupt_flag_get
Function prototype	FlagStatus pkcau_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get PKCAU interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	

<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>	
<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 3-401. 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);
```

## 3.16. PMU

According to the Power management unit (PMU), provides six types of power saving modes, including Sleep, Deep-sleep, Standby mode, SRAM\_sleep, Wi-Fi\_sleep and BLE\_sleep mode. The PMU registers are listed in chapter [3.16.1](#), the PMU firmware functions are introduced in chapter [3.16.2](#).

### 3.16.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

**Table 3-402. PMU Registers**

Registers	Descriptions
PMU_CTL0	PMU control register 0
PMU_CS0	PMU control and status register 0
PMU_CTL1	PMU control register 1
PMU_CS1	PMU control and status register 1
PMU_PAR0	PMU parameter register 0
PMU_PAR1	PMU parameter register 1
PMU_PAR2	PMU parameter register 2
PMU_RFCTL	PMU RF control register
PMU_RFPAR	PMU RF timer parameter register
PMU_INTF	PMU BLE interrupt flag register
PMU_INTEN	PMU BLE interrupt enable register
PMU_INTC	PMU BLE interrupt clear register

### 3.16.2. Descriptions of Peripheral functions

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

**Table 3-403. PMU firmware function**

Function name	Function description
pmu_deinit	reset PMU register
pmu_lvd_select	select low voltage detector threshold
pmu_lvd_disable	disable PMU LVD
pmu_backup_write_enable	enable write access to the registers in backup domain
pmu_backup_write_disable	disable write access to the registers in backup domain
pmu_to_sleepmode	PMU work in sleep mode
pmu_to_deepsleepmode	PMU work in deep-sleep mode
pmu_to_standbymode	PMU work in standby mode
pmu_wakeup_pin_enable	enable wakeup pin
pmu_wakeup_pin_disable	disable wakeup pin

Function name	Function description
pmu_wifi_power_enable	enable WIFI power
pmu_wifi_power_disable	disable WIFI power
pmu_wifi_sram_control	WIFI & SRAM low power control
pmu_ble_control	BLE low power control
pmu_ble_wakeup_request_enable	enable BLE wakeup request
pmu_ble_wakeup_request_disable	disable BLE wakeup request
pmu_pll_force_enable	MCU PLL power force open/close
pmu_pll_force_disable	disable MCU PLL power open/close force
pmu_ble_rf_config	configure BLE RF sequence
pmu_rf_force_enable	enable RF sequence force open/close
pmu_rf_force_disable	disable RF sequence open/close force
pmu_rf_sequence_config	configure RF sequence
pmu_flag_get	get flag status
pmu_flag_clear	clear flag bit
pmu_interrupt_enable	enable PMU interrupt
pmu_interrupt_disable	disable PMU interrupt
pmu_interrupt_flag_get	get PMU interrupt flag
pmu_interrupt_flag_clear	clear PMU interrupt flag

## pmu\_deinit

The description of pmu\_deinit is shown as below:

**Table 3-404. Function pmu\_deinit**

<b>Function name</b>	pmu_deinit
<b>Function prototype</b>	void pmu_deinit(void);
<b>Function descriptions</b>	reset PMU register
<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 PMU */

pmu_deinit();

```

## pmu\_lvd\_select

The description of pmu\_lvd\_select is shown as below:

**Table 3-405. Function pmu\_lvd\_select**

<b>Function name</b>	pmu_lvd_select
<b>Function prototype</b>	void pmu_lvd_select(uint32_t lvd_t_n);
<b>Function descriptions</b>	select low voltage detector threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lvd_t_n</b>	voltage threshold value
<i>PMU_LVDT_0</i>	voltage threshold is 2.1V
<i>PMU_LVDT_1</i>	voltage threshold is 2.3V
<i>PMU_LVDT_2</i>	voltage threshold is 2.4V
<i>PMU_LVDT_3</i>	voltage threshold is 2.6V
<i>PMU_LVDT_4</i>	voltage threshold is 2.7V
<i>PMU_LVDT_5</i>	voltage threshold is 2.9V
<i>PMU_LVDT_6</i>	voltage threshold is 3.0V
<i>PMU_LVDT_7</i>	voltage threshold is 3.1V
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select low voltage detector threshold as 2.9V */
```

```
pmu_lvd_select(PMU_LVDT_5);
```

## pmu\_lvd\_disable

The description of pmu\_lvd\_disable is shown as below:

**Table 3-406. 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\_backup\_write\_enable

The description of pmu\_backup\_write\_enable is shown as below:

**Table 3-407. 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 write access to the registers in backup domain
<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 the registers in backup domain */
```

```
pmu_backup_write_enable();
```

### pmu\_backup\_write\_disable

The description of pmu\_backup\_write\_disable is shown as below:

**Table 3-408. 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 write access to the registers in backup domain
<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 the registers in backup domain */
```

```
pmu_backup_write_disable();
```

## pmu\_to\_sleepmode

The description of pmu\_to\_sleepmode is shown as below:

**Table 3-409. Function pmu\_to\_sleepmode**

<b>Function name</b>	pmu_to_sleepmode
<b>Function prototype</b>	void pmu_to_sleepmode(uint8_t sleepmodecmd);
<b>Function descriptions</b>	PMU work in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sleepmodecmd</b>	command to enter sleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-410. 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 in deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ldo</b>	ldo work mode
<i>PMU_LDO_NORMAL</i>	LDO operates normally 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>	Low-driver mode
<i>PMU_LOWDRIVER_DISABLE</i>	Low-driver mode disable in deep-sleep mode
<i>PMU_LOWDRIVER_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 in deep-sleep mode */
```

```
pmu_to_deepsleepmode(PMU_LDO_NORMAL, PMU_LOWDRIVER_DISABLE, WFI_CMD);
```

### pmu\_to\_standbymode

The description of pmu\_to\_standbymode is shown as below:

**Table 3-411. Function pmu\_to\_standbymode**

<b>Function name</b>	pmu_to_standbymode
<b>Function prototype</b>	void pmu_to_standbymode(uint8_t standbymodecmd);
<b>Function descriptions</b>	PMU work in standby mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>standbymodecmd</b>	command to enter standby mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PMU work at standby mode */
```

```
pmu_to_standbymode(WFI_CMD);
```

## pmu\_wakeup\_pin\_enable

The description of pmu\_wakeup\_pin\_enable is shown as below:

**Table 3-412. Function pmu\_wakeup\_pin\_enable**

<b>Function name</b>	pmu_wakeup_pin_enable
<b>Function prototype</b>	void pmu_wakeup_pin_enable(void);
<b>Function descriptions</b>	enable wakeup pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_pin</b>	WKUP pin
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PA15)
PMU_WAKEUP_PIN2	WKUP Pin 2 (PA7)
PMU_WAKEUP_PIN3	WKUP Pin 3 (PA12)
<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 3-413. Function pmu\_wakeup\_pin\_disable**

<b>Function name</b>	pmu_wakeup_pin_disable
<b>Function prototype</b>	void pmu_wakeup_pin_disable(void);
<b>Function descriptions</b>	disable wakeup pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_pin</b>	WKUP pin
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PA15)
PMU_WAKEUP_PIN2	WKUP Pin 2 (PA7)
PMU_WAKEUP_PIN3	WKUP Pin 3 (PA12)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* disable wakeup pin 0 */
```

```
pmu_wakeup_pin_disable(PMU_WAKEUP_PIN0);
```

### pmu\_wifi\_power\_enable

The description of pmu\_wifi\_power\_enable is shown as below:

**Table 3-414. 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>	enable WIFI power
<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 3-415. 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>	disable WIFI power
<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\_sram\_control

The description of pmu\_wifi\_sram\_control is shown as below:

**Table 3-416. Function pmu\_wifi\_sram\_control**

<b>Function name</b>	pmu_wifi_sram_control
<b>Function prototype</b>	void pmu_wifi_sram_control(uint32_t wifi_sram);
<b>Function descriptions</b>	WIFI & SRAM low power control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wakeup_sram</b>	low power control
<i>PMU_WIFI_SLEEP</i>	WIFI go to sleep
<i>PMU_WIFI_WAKE</i>	WIFI wakeup
<i>PMU_SRAM1_SLEEP</i>	SRAM1 go to sleep
<i>PMU_SRAM1_WAKE</i>	SRAM1 wakeup
<i>PMU_SRAM2_SLEEP</i>	SRAM2 go to sleep
<i>PMU_SRAM2_WAKE</i>	SRAM2 wakeup
<i>PMU_SRAM3_SLEEP</i>	SRAM3 go to sleep
<i>PMU_SRAM3_WAKE</i>	SRAM3 wakeup
<i>PMU_SRAM0_SLEEP</i>	SRAM0 go to sleep
<i>PMU_SRAM0_WAKE</i>	SRAM0 wakeup
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* WIFI & SRAM low power control */
pmu_wifi_sram_control(PMU_WIFI_SLEEP);
```

### pmu\_ble\_control

The description of pmu\_ble\_control is shown as below:

**Table 3-417. Function pmu\_ble\_control**

<b>Function name</b>	pmu_ble_control
<b>Function prototype</b>	void pmu_ble_control(uint32_t wifi_sram);
<b>Function descriptions</b>	BLE low power control

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ble</b>	BLE low power control
<i>PMU_BLE_SLEEP</i>	BLE go to sleep
<i>PMU_BLE_WAKE</i>	BLE wakeup
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* BLE low power control */
pmu_ble_control(PMU_BLE_SLEEP);
```

### pmu\_ble\_wakeup\_request\_enable

The description of pmu\_ble\_wakeup\_request\_enable is shown as below:

**Table 3-418. Function pmu\_ble\_wakeup\_request\_enable**

<b>Function name</b>	pmu_ble_wakeup_request_enable
<b>Function prototype</b>	void pmu_ble_wakeup_request_enable(void);
<b>Function descriptions</b>	enable BLE wakeup request
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable BLE wakeup request */
pmu_ble_wakeup_request_enable();
```

### pmu\_ble\_wakeup\_request\_disable

The description of pmu\_ble\_wakeup\_request\_disable is shown as below:

**Table 3-419. Function pmu\_ble\_wakeup\_request\_disable**

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

<b>Function descriptions</b>	disable BLE wakeup request
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable BLE wakeup request */
pmu_ble_wakeup_request_disable();
```

### pmu\_pll\_force\_enable

The description of pmu\_pll\_force\_enable is shown as below:

**Table 3-420. Function pmu\_pll\_force\_enable**

<b>Function name</b>	pmu_pll_force_enable
<b>Function prototype</b>	void pmu_pll_force_enable(uint32_t force);
<b>Function descriptions</b>	enable MCU PLL power force open/close
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>force</b>	PLL power force open/close
<i>PMU_PLL_FORCE_OPEN</i>	Software force open, open MCU PLL power
<i>PMU_PLL_FORCE_CLOSE</i>	Software force close, close MCU PLL power
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable MCU PLL power force open/close */
pmu_pll_force_enable(PMU_PLL_FORCE_OPEN);
```

### pmu\_pll\_force\_disable

The description of pmu\_pll\_force\_disable is shown as below:

Table 3-421. Function `pmu_pll_force_disable`

<b>Function name</b>	<code>pmu_pll_force_disable</code>
<b>Function prototype</b>	<code>void pmu_pll_force_disable(uint32_t force);</code>
<b>Function descriptions</b>	disable MCU PLL power open/close force
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>force</b>	PLL power force open/close
<code>PMU_PLL_FORCE_OPEN</code>	Software force open, open MCU PLL power
<code>PMU_PLL_FORCE_CLOSE</code>	Software force close, close MCU PLL power
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable MCU PLL power force open/close */
pmu_pll_force_disable(PMU_PLL_FORCE_CLOSE);
```

### **`pmu_ble_rf_config`**

The description of `pmu_ble_rf_config` is shown as below:

Table 3-422. Function `pmu_ble_rf_config`

<b>Function name</b>	<code>pmu_ble_rf_config</code>
<b>Function prototype</b>	<code>void pmu_ble_rf_config(uint32_t mode);</code>
<b>Function descriptions</b>	configure BLE RF sequence
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	enable RF mode
<code>PMU_BLE_RF_SOFTWARE</code>	enable RF by software
<code>PMU_BLE_RF_HARDWARE</code>	enable RF BLE hardware
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable RF by software */
```

```
pmu_ble_rf_config(PMU_BLE_RF_SOFTWARE);
```

### pmu\_rf\_force\_enable

The description of pmu\_rf\_force\_enable is shown as below:

**Table 3-423. 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>	RF sequence force open/close
<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 RF sequence force open/close */
```

```
pmu_rf_force_enable(PMU_RF_FORCE_OPEN);
```

### pmu\_rf\_force\_disable

The description of pmu\_rf\_force\_disable is shown as below:

**Table 3-424. 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>	RF sequence force open/close
<i>PMU_RF_FORCE_OPEN</i>	Software force start, open RF power
<i>PMU_RF_FORCE_CLOSE</i>	Software force close, close RF power

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

Example:

```
/* disable RF sequence open/close force */
pmu_rf_force_disable(PMU_RF_FORCE_OPEN);
```

### pmu\_rf\_sequence\_config

The description of pmu\_rf\_sequence\_config is shown as below:

**Table 3-425. Function pmu\_rf\_sequence\_config**

Function name	pmu_rf_sequence_config
Function prototype	void pmu_rf_sequence_config(uint32_t mode);
Function descriptions	configure RF sequence
Precondition	-
The called functions	-
Input parameter{in}	
mode	RF sequence mode
PMU_RF_SOFTWARE	RF software sequence enable
PMU_RF_HARDWARE	RF hardware sequence enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure RF sequence */
pmu_rf_sequence_config(PMU_RF_SOFTWARE);
```

### pmu\_flag\_get

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

**Table 3-426. Function pmu\_flag\_get**

Function name	pmu_flag_get
Function prototype	FlagStatus pmu_flag_get(uint32_t flag);
Function descriptions	get flag state
Precondition	-
The called functions	-

Input parameter{in}	
flag	flag
PMU_FLAG_WAKEUP	wakeup flag
PMU_FLAG_STANDBY	standby flag
PMU_FLAG_LVD	lvd 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_BLE_POWER	BLE power status
PMU_FLAG_SRAM1_SLEEP	SRAM1 is in sleep state
PMU_FLAG_SRAM1_ACTIVE	SRAM1 is in active state
PMU_FLAG_SRAM0_SLEEP	SRAM0 is in sleep state
PMU_FLAG_SRAM0_ACTIVE	SRAM0 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
PMU_FLAG_BLE_SLEEP	BLE is in sleep state
PMU_FLAG_BLE_ACTIVE	BLE is in active state
PMU_FLAG_BLE_POWER_FALL	BLE power status falling edge flag
PMU_FLAG_BLE_POWER_RISE	BLE power status rising edge flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get flag state */
```

```
FlagStatus status;
```

```
status = pmu_flag_get(PMU_FLAG_WAKEUP);
```

## pmu\_flag\_clear

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

**Table 3-427. 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
PMU_FLAG_RESET_WAKEUP	reset wakeup flag
PMU_FLAG_RESET_STANDBY	reset standby flag
PMU_FLAG_RESET_LDRF	reset Low-driver mode ready flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear flag bit */
```

```
pmu_flag_clear(PMU_FLAG_RESET_WAKEUP);
```

## pmu\_interrupt\_enable

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

**Table 3-428. Function pmu\_interrupt\_enable**

<b>Function name</b>	pmu_interrupt_enable
<b>Function prototype</b>	void pmu_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable PMU interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	PMU interrupt

<i>PMU_INT_BLE_POWER_FALL</i>	BLE power status falling edge interrupt
<i>PMU_INT_BLE_POWER_RISE</i>	BLE power status rising edge interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable PMU interrupt */

pmu_interrupt_enable(PMU_INT_BLE_POWER_FALL);
```

### pmu\_interrupt\_disable

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

**Table 3-429. Function pmu\_interrupt\_disable**

<b>Function name</b>	pmu_interrupt_disable
<b>Function prototype</b>	void pmu_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable PMU interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	PMU interrupt
<i>PMU_INT_BLE_POWER_FALL</i>	BLE power status falling edge interrupt
<i>PMU_INT_BLE_POWER_RISE</i>	BLE power status rising edge interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable PMU interrupt */

pmu_interrupt_disable(PMU_INT_BLE_POWER_FALL);
```

### pmu\_interrupt\_flag\_get

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

**Table 3-430. Function pmu\_interrupt\_flag\_get**

<b>Function name</b>	pmu_interrupt_flag_get
----------------------	------------------------

<b>Function prototype</b>	FlagStatus pmu_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	get PMU interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	PMU interrupt flag
<i>PMU_INT_FLAG_BLE_POWER_FALL</i>	BLE power status falling edge interrupt
<i>PMU_INT_FLAG_BLE_POWER_RISE</i>	BLE power status rising edge interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* get PMU interrupt flag */
```

```
pmu_interrupt_flag_get(PMU_INT_FLAG_BLE_POWER_FALL);
```

### pmu\_interrupt\_flag\_clear

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

**Table 3-431. Function pmu\_interrupt\_flag\_clear**

<b>Function name</b>	pmu_interrupt_flag_clear
<b>Function prototype</b>	FlagStatus pmu_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear PMU interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	PMU interrupt flag
<i>PMU_INT_FLAG_RESET_BLE_POWER_FALL</i>	BLE power status falling edge interrupt
<i>PMU_INT_FLAG_RESET_BLE_POWER_RISE</i>	BLE power status rising edge interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear PMU interrupt flag */
```

```
pmu_interrupt_flag_get(PMU_INT_FLAG_RESET_BLE_POWER_FALL);
```

## 3.17. 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 [3.17.1](#), the QSPI firmware functions are introduced in chapter [3.17.2](#).

### 3.17.1. Descriptions of Peripheral registers

QSPI registers are listed in the table shown as below:

**Table 3-432. 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

### 3.17.2. Descriptions of Peripheral functions

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

**Table 3-433. QSPI firmware function**

Function name	Function description
qspi_deinit	reset QSPI peripheral
qspi_struct_para_init	initialize the parameters of QSPI init structure with the default values
qspi_cmd_struct_para_init	initialize the parameters of QSPI command structure with the default values
qspi_polling_struct_para_init	initialize the parameters of QSPI read polling structure with the default values

Function name	Function description
qspi_init	initialize QSPI
qspi_enable	enable QSPI
qspi_disable	disable QSPI
qspi_dma_enable	enable QSPI DMA
qspi_dma_disable	disable QSPI DMA
qspi_command_config	configure QSPI command parameters
qspi_polling_config	configure QSPI read polling mode
qspi_memorymapped_config	configure QSPI memory mapped mode
qspi_data_transmit	QSPI transmit data function
qspi_data_receive	QSPI receive data function
qspi_transmission_abort	abort the current transmission
qspi_flag_get	get QSPI flag status
qspi_flag_clear	clear QSPI flag status
qspi_interrupt_enable	enable QSPI interrupt
qspi_interrupt_disable	disable QSPI interrupt
qspi_interrupt_flag_get	get QSPI interrupt flag status
qspi_interrupt_flag_clear	clear QSPI interrupt flag status

### Structure qspi\_init\_struct

**Table 3-434. Structure qspi\_init\_struct**

Member name	Function description
prescaler	QSPI prescaler
fifo_threshold	QSPI FIFO threshold
sample_shift	QSPI sample shift
flash_size	external flash size
cs_high_time	CLK cycles which the chip select (nCS) must stay high between two command sequences
clock_mode	QSPI clock mode

### Structure qspi\_command\_struct

**Table 3-435. Structure qspi\_command\_struct**

Member name	Function description
instruction_mode	instruction mode
instruction	8 bits instruction
addr_mode	address mode
addr_size	address size
addr	address to be send to the external flash memory
altebytes_mode	alternate bytes mode
altebytes_size	alternate bytes size
altebytes	alternate bytes information

Member name	Function description
dummycycles	dummy cycles
data_mode	data mode
data_length	32 bits data length
sioo_mode	send instruction only once mode

### Structure qspi\_polling\_struct

**Table 3-436. Structure qspi\_polling\_struct**

Member name	Function description
match	match value
mask	mask value
interval	number of clock cycles between two read during automatic polling phases
statusbytes_size	the size of the status bytes received
match_mode	method used for determining a match
polling_stop	if read polling is stopped after a match

### qspi\_deinit

The description of qspi\_deinit is shown as below:

**Table 3-437. 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\_struct\_para\_init

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

**Table 3-438. Function qspi\_struct\_para\_init**

<b>Function name</b>	qspi_struct_para_init
----------------------	-----------------------

<b>Function prototype</b>	void qspi_struct_para_init(qspi_init_struct *init_para);
<b>Function descriptions</b>	initialize the parameters of QSPI structure with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>init_para</b>	QSPI parameter structure, the structure members can refer to <a href="#">Table 3-434. Structure qspi_init_struct.</a>
<b>Return value</b>	
-	-

Example:

```
/* initialize the parameters of QSPI */
qspi_init_struct qspi_init_para;
qspi_struct_para_init(&qspi_init_para);
```

### qspi\_cmd\_struct\_para\_init

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

**Table 3-439. Function qspi\_cmd\_struct\_para\_init**

<b>Function name</b>	qspi_cmd_struct_para_init
<b>Function prototype</b>	void qspi_cmd_struct_para_init(qspi_command_struct *init_para);
<b>Function descriptions</b>	initialize the parameters of QSPI command structure with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>init_para</b>	QSPI command parameter structure, the structure members can refer to <a href="#">Table 3-435. Structure qspi_command_struct.</a>
<b>Return value</b>	
-	-

Example:

```
/* initialize the parameters of QSPI command structrue */
qspi_command_struct qspi_cmd_para;
qspi_cmd_struct_para_init(&qspi_cmd_para);
```

## qspi\_polling\_struct\_para\_init

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

**Table 3-440. Function qspi\_polling\_struct\_para\_init**

<b>Function name</b>	qspi_polling_struct_para_init
<b>Function prototype</b>	void qspi_polling_struct_para_init(qspi_polling_struct *init_para);
<b>Function descriptions</b>	initialize the parameters of QSPI read polling structure with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
init_para	QSPI read polling parameter structure, the structure members can refer to <a href="#">Table 3-436. Structure qspi_polling_struct.</a>
<b>Return value</b>	
-	-

Example:

```
/* initialize the parameters of QSPI read polling structrue */
```

```
qspi_polling_struct qspi_polling_para;
```

```
qspi_polling_struct_para_init(&qspi_polling_para);
```

## qspi\_init

The description of qspi\_init is shown as below:

**Table 3-441. 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
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
qspi_struct	QSPI parameter initialization structure, the structure members can refer to members of the structure <a href="#">Table 3-434. Structure qspi_init_struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the init parameter structure */
```

```
qspi_struct_para_init(&qspi_init_para);
```

```
qspi_init_para.clock_mode = QSPI_CLOCK_MODE_0;
```

```
qspi_init_para.fifo_threshold = 4;
```

```
qspi_init_para.sample_shift = QSPI_SAMPLE_SHIFTING_HALFCYCLE;
```

```
qspi_init_para.cs_high_time = QSPI_CS_HIGH_TIME_2_CYCLE;
```

```
qspi_init_para.flash_size = 27;
```

```
qspi_init_para.prescaler = 4;
```

```
qspi_init(&qspi_init_para);
```

### qspi\_enable

The description of qspi\_enable is shown as below:

**Table 3-442. Function qspi\_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 3-443. 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>	-

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

Example:

```
/* disable QSPI */
```

```
qspi_disable();
```

### qspi\_dma\_enable

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

**Table 3-444. Function qspi\_dma\_enable**

Function name	qspi_dma_enable
Function prototype	void qspi_dma_enable(void);
Function descriptions	enable QSPI DMA
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable QSPI DMA */
```

```
qspi_dma_enable();
```

### qspi\_dma\_disable

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

**Table 3-445. Function qspi\_dma\_disable**

Function name	qspi_dma_disable
Function prototype	void qspi_dma_disable(void);
Function descriptions	disable QSPI DMA
Precondition	-
The called functions	-

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

Example:

```
/* disable QSPI DMA */
```

```
qspi_dma_disable();
```

### qspi\_command\_config

The description of qspi\_command\_config is shown as below:

**Table 3-446. Function qspi\_command\_config**

Function name	qspi_command_config
Function prototype	void qspi_command_config(qspi_command_struct *cmd);
Function descriptions	configure QSPI command parameters
Precondition	-
The called functions	-
Input parameter{in}	
cmd	QSPI command parameter structure, the structure members can refer to <a href="#">Table 3-435. Structure qspi_command_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the QSPI command parameter structure */
```

```
qspi_cmd_struct_para_init(&qspi_cmd);
```

```
/* write enable */
```

```
qspi_cmd.instruction_mode = QSPI_INSTRUCTION_1_LINE;
```

```
qspi_cmd.instruction = WRITE_ENABLE_CMD;
```

```
qspi_cmd.addr_mode = QSPI_ADDR_NONE;
```

```
qspi_cmd.altebytes_mode = QSPI_ALTE_BYTES_NONE;
```

```
qspi_cmd.data_mode = QSPI_DATA_NONE;
```

```
qspi_cmd.dummycycles = 0;
```

```
qspi_cmd.sioo_mode = QSPI_SIOO_INST_EVERY_CMD;
```

```
qspi_command_config(&qspi_cmd);
```

### qspi\_polling\_config

The description of qspi\_polling\_config is shown as below:

**Table 3-447. Function qspi\_polling\_config**

Function name	qspi_polling_config
Function prototype	void qspi_polling_config(qspi_command_struct *cmd, qspi_polling_struct *cfg);
Function descriptions	configure QSPI read polling mode
Precondition	-
The called functions	-
Input parameter{in}	
cmd	QSPI command parameter structure, the structure members can refer to <a href="#">Table 3-435. Structure qspi_command_struct</a> .
Input parameter{in}	
cfg	QSPI read polling parameter structure, the structure members can refer to <a href="#">Table 3-436. Structure qspi_polling_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the QSPI command parameter structure */
qspi_cmd_struct_para_init(&qspi_cmd);

/* initialize the QSPI read polling parameter structure */
qspi_polling_struct_para_init(&polling_cmd);

/* read enable */
polling_cmd.match = 0x02;
polling_cmd.mask = 0x02;
polling_cmd.match_mode = QSPI_MATCH_MODE_AND;
polling_cmd.statusbytes_size = 1;
polling_cmd.interval = 0x10;
polling_cmd.polling_stop = QSPI_POLLING_STOP_ENABLE;
qspi_cmd.instruction = READ_STATUS_REG1_CMD;
```

```

qspi_cmd.data_mode = QSPI_DATA_1_LINE;

qspi_polling_config(&qspi_cmd, &polling_cmd);

```

## qspi\_memorymapped\_config

The description of qspi\_memorymapped\_config is shown as below:

**Table 3-448. Function qspi\_memorymapped\_config**

<b>Function name</b>	qspi_memorymapped_config
<b>Function prototype</b>	void qspi_memorymapped_config(qspi_command_struct *cmd, uint16_t timeout, uint32_t toen);
<b>Function descriptions</b>	configure QSPI memory mapped mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmd</b>	QSPI command parameter structure, the structure members can refer to <a href="#">Table 3-435. Structure qspi_command_struct</a> .
<b>Input parameter{in}</b>	
<b>timeout</b>	0-0xFFFF
<b>Input parameter{in}</b>	
<b>toen</b>	timeout counter
QSPI_TMOUT_DISABLE	disable timeout counter
QSPI_TMOUT_ENABLE	enable timeout counter
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* initialize the QSPI command parameter structure */

qspi_cmd_struct_para_init(&qspi_cmd);

qspi_cmd.instruction_mode = QSPI_INSTRUCTION_1_LINE;

qspi_cmd.instruction = RDID;

qspi_cmd.addr_mode = QSPI_ADDR_NONE;

qspi_cmd.addr_size = QSPI_ADDR_8_BITS;

qspi_cmd.addr = 0;

qspi_cmd.altebytes_mode = QSPI_ALTE_BYTES_1_LINE;

```

```

qspi_cmd.altebytes_size = QSPI_ALTE_BYTES_24_BITS;

qspi_cmd.altebytes = 0;

qspi_cmd.dummycycles = 0;

qspi_cmd.data_mode = QSPI_DATA_1_LINE;

qspi_cmd.data_length = 3;

qspi_cmd.sioo_mode = QSPI_SIOO_INST_EVERY_CMD;

qspi_memorymapped_config(&qspi_cmd, 0xf, QSPI_TMOUT_ENABLE);

```

### qspi\_data\_transmit

The description of qspi\_data\_transmit is shown as below:

**Table 3-449. Function qspi\_data\_transmit**

<b>Function name</b>	qspi_data_transmit
<b>Function prototype</b>	void qspi_data_transmit(uint8_t *tdata);
<b>Function descriptions</b>	QSPI transmit data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>tdata</b>	pointer to the data to be transmitted
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* QSPI transmit data */

uint8_t data[32];

qspi_data_transmit(&data);

```

### qspi\_data\_receive

The description of qspi\_data\_receive is shown as below:

**Table 3-450. Function qspi\_data\_receive**

<b>Function name</b>	qspi_data_receive
<b>Function prototype</b>	void qspi_data_transmit(uint8_t *tdata);
<b>Function descriptions</b>	QSPI transmit data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>rdata</b>	pointer to the data to be received
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* QSPI receive data */
uint8_t data[32];
qspi_data_receive(&data);
```

### qspi\_transmission\_abort

The description of qspi\_transmission\_abort is shown as below:

**Table 3-451. Function qspi\_transmission\_abort**

<b>Function name</b>	qspi_transmission_abort
<b>Function prototype</b>	void qspi_transmission_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 transmission */
qspi_transmission_abort();
```

### qspi\_flag\_get

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

**Table 3-452. 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
<i>QSPI_FLAG_BUSY</i>	busy flag
<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_RPMF</i>	read polling match flag
<i>QSPI_FLAG_TMOUT</i>	timeout flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get QSPI transfer complete flag */
```

```
FlagStatus status;
```

```
status = qspi_flag_get(QSPI_FLAG_TC);
```

### qspi\_flag\_clear

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

**Table 3-453. 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
<i>QSPI_FLAG_TERR</i>	transfer error flag
<i>QSPI_FLAG_TC</i>	transfer complete flag
<i>QSPI_FLAG_RPMF</i>	read polling match flag
<i>QSPI_FLAG_TMOUT</i>	timeout flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear QSPI transfer complete flag status */
```

```
qspi_flag_clear(QSPI_FLAG_TC);
```

## qspi\_interrupt\_enable

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

**Table 3-454. Function qspi\_interrupt\_enable**

<b>Function name</b>	qspi_interrupt_enable
<b>Function prototype</b>	void qspi_interrupt_enable(uint32_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_RPMF	read polling match interrupt
QSPI_INT_TMOUT	timeout 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 3-455. 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_RPMF	read polling match interrupt
QSPI_INT_TMOUT	timeout interrupt
<b>Output parameter{out}</b>	

-	-
Return value	

Example:

```
/* disable QSPI transfer complete interrupt */
qspi_interrupt_disable(QSPI_INT_TC);
```

### qspi\_interrupt\_flag\_get

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

**Table 3-456. Function qspi\_interrupt\_flag\_get**

<b>Function name</b>	qspi_interrupt_flag_get
<b>Function prototype</b>	FlagStatus qspi_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	get QSPI interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	QSPI interrupt flag
QSPI_INT_FLAG_TER R	transfer error interrupt flag
QSPI_INT_FLAG_TC	transfer complete interrupt flag
QSPI_INT_FLAG_FT	FIFO threshold interrupt flag
QSPI_INT_FLAG_RPM F	read polling match interrupt flag
QSPI_INT_FLAG_TMO UT	timeout interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get QSPI transfer complete flag */
FlagStatus status;
status = qspi_flag_get(QSPI_FLAG_TC);
```

### qspi\_interrupt\_flag\_clear

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

Table 3-457. Function `qspi_interrupt_flag_clear`

<b>Function name</b>	<code>qspi_interrupt_flag_clear</code>
<b>Function prototype</b>	<code>void qspi_interrupt_flag_clear(uint32_t int_flag);</code>
<b>Function descriptions</b>	clear QSPI interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	QSPI interrupt flag
<code>QSPI_INT_FLAG_TER</code> <i>R</i>	transfer error interrupt flag
<code>QSPI_INT_FLAG_TC</code>	transfer complete interrupt flag
<code>QSPI_INT_FLAG_RPM</code> <i>F</i>	read polling match interrupt flag
<code>QSPI_INT_FLAG_TMO</code> <i>UT</i>	timeout interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear QSPI transfer complete flag status */
qspi_flag_clear(QSPI_FLAG_TC);
```

## 3.18. 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 [3.18.1](#), the RCU firmware functions are introduced in chapter [3.18.2](#).

### 3.18.1. Descriptions of Peripheral registers

RCU registers are listed in the table shown as below:

Table 3-458. RCU Registers

Registers	Descriptions
RCU_CTL	Control register
RCU_PLL	PLL register
RCU_CFG0	Clock configuration register 0
RCU_INT	Clock interrupt register
RCU_AHB1RST	AHB1 reset register

Registers	Descriptions
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_BDCTL	Backup domain control register
RCU_RSTSCK	Reset source/clock register
RCU_PLLDIGCFG0	PLLDIG clock configuration register 0
RCU_CFG1	Clock configuration register 1
RCU_ADDCTL	Additional clock control register
RCU_PLLDIGCFG1	PLLDIG clock configuration register 1
RCU_VKEY	Voltage key register
RCU_DSV	Deep-sleep mode voltage register

### 3.18.2. Descriptions of Peripheral functions

**Table 3-459. RCU firmware function**

Function name	Function description
rcu_deinit	deinitialize the RCU
rcu_irc16m_dfs_to_rf_enable	enable differential signal of IRC16M to RF module
rcu_irc16m_dfs_to_rf_disable	disable differential signal of IRC16M to RF module
rcu_periph_clock_enable	enable the peripherals clock
rcu_periph_clock_disable	disable the peripherals clock
rcu_fmc_clock_sleep_enable	enabled FMC clock when sleep mode
rcu_fmc_clock_sleep_disable	disabled FMC clock when sleep mode
rcu_periph_reset_enable	enable the peripherals reset
rcu_periph_reset_disable	disable the peripheral reset
rcu_bkp_reset_enable	enable the BKP domain reset
rcu_bkp_reset_disable	disable the BKP domain reset
rcu_rfpll_cal_enable	enable the RF PLL calculation
rcu_rfpll_cal_disable	disable the RF PLL calculation
rcu_control_unit_powerup	power up rcu control unit
rcu_control_unit_powerdown	power down rcu control unit
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

Function name	Function description
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 and divider
rcu_ckout1_config	configure the CK_OUT1 clock source and divider
rcu_plldig_config	configure the PLLDIG output clock frequency
rcu_plldigdiv_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_trng_div_config	configure the frequency division of the TRNG source clock
rcu_i2c0_clock_config	configure the I2C0 clock source selection
rcu_usart0_clock_config	configure the USART0 clock source selection
rcu_irc16m_div_config	configure IRC16M clock divider factor for system clock
rcu_timer_clock_prescaler_config	configure the TIMER clock prescaler selection
rcu_flag_get	get the clock stabilization and peripheral reset flags
rcu_all_reset_flag_clear	clear all the reset flag
rcu_interrupt_flag_get	get the clock stabilization interrupt and ckm flags
rcu_interrupt_flag_clear	clear the interrupt flags
rcu_interrupt_enable	enable the stabilization interrupt
rcu_interrupt_disable	disable the stabilization interrupt
rcu_lxtal_drive_capability_config	configure the LXTAL drive capability
rcu_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_rf_hxtal_clock_monitor_enable	enable the RF HXTAL clock monitor
rcu_rf_hxtal_clock_monitor_disable	disable the RF HXTAL clock monitor
rcu_irc16m_adjust_value_set	set the IRC16M adjust value
rcu_voltage_key_unlock	unlock the voltage key
rcu_deepsleep_voltage_set	select deep-sleep mode voltage
rcu_clock_freq_get	get the system clock, bus clock frequency

## Enum rcu\_periph\_enum

**Table 3-460. Enum rcu\_periph\_enum**

enum name	Function description
RCU_GPIOA	GPIOA clock
RCU_GPIOB	GPIOB clock
RCU_GPIOC	GPIOC clock
RCU_CRC	CRC clock

enum name	Function description
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_DMA	DMA clock
RCU_BLE	BLE clock
RCU_PKCAU	PKCAU clock
RCU_CAU	CAU clock
RCU_HAU	HAU clock
RCU_TRNG	TRNG clock
RCU_QSPI	QSPI clock
RCU_TIMER1	TIMER1 clock
RCU_TIMER2	TIMER2 clock
RCU_TIMER5	TIMER5 clock
RCU_WWDGT	WWDGT clock
RCU_RFI	RFI clock
RCU_UART1	UART1 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_UART2	UART0 clock
RCU_ADC	ADC clock
RCU_SPI	SPI clock
RCU_SYSCFG	SYSCFG clock
RCU_TIMER15	TIMER15 clock
RCU_TIMER16	TIMER16 clock
RCU_RF	RF clock

## Enum rcu\_periph\_reset\_enum

Table 3-461. 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_CRCRST	CRC clock reset
RCU_WIFIRST	WIFI clock reset

enum name	Function description
RCU_DMARST	DMA clock reset
RCU_BLERST	BLE clock reset
RCU_PKCAURST	PKCAU clock reset
RCU_CAURST	CAU clock reset
RCU_HAURST	HAU clock reset
RCU_TRNGRST	TRNG clock reset
RCU_QSPIRST	QSPI clock reset
RCU_TIMER1RST	TIMER1 clock reset
RCU_TIMER2RST	TIMER2 clock reset
RCU_TIMER5RST	TIMER5 clock reset
RCU_WWDGTRST	WWDGT clock reset
RCU_RFIRST	RFI clock reset
RCU_UART1RST	UART1 clock reset
RCU_USART0RST	USART0 clock reset
RCU_I2C0RST	I2C0 clock reset
RCU_I2C1RST	I2C1 clock reset
RCU_PMURST	PMU clock reset
RCU_TIMER0RST	TIMER0 clock reset
RCU_UART2RST	UART2 clock reset
RCU_ADCRST	ADC clock reset
RCU_SPIRST	SPI clock reset
RCU_SYSCFGRST	SYSCFG clock reset
RCU_TIMER15RST	TIMER15 clock reset
RCU_TIMER16RST	TIMER16 clock reset
RCU_RFRST	RF clock reset

### Enum rcu\_unit\_enum

Table 3-462. 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

### Enum rcu\_flag\_enum

Table 3-463. Enum rcu\_flag\_enum

enum name	Function description
RCU_FLAG_IRC16MS	IRC16M stabilization flags

enum name	Function description
TB	
RCU_FLAG_HXTALST B	HXTAL stabilization flags
RCU_FLAG_PLLDIGST B	PLLDIG stabilization flags
RCU_FLAG_LXTALST B	LXTAL stabilization flags
RCU_FLAG_IRC32KST B	IRC32K 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 3-464. 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_PLLD IGSTB	PLLDIG stabilization interrupt flag
RCU_INT_FLAG_CKM	HXTAL clock stuck interrupt flag

### Enum rcu\_int\_flag\_clear\_enum

**Table 3-465. 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

enum name	Function description
RCU_INT_FLAG_IRC16MSTB_CLR	IRC16M stabilization interrupt flags clear
RCU_INT_FLAG_HXTAL_LSTB_CLR	HXTAL stabilization interrupt flags clear
RCU_INT_FLAG_PLLDIGSTB_CLR	PLLDIG stabilization interrupt flags clear
RCU_INT_FLAG_CKM_CLR	CKM interrupt flags clear

### Enum rcu\_int\_enum

Table 3-466. 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_PLLDIGSTB	PLLDIG stabilization interrupt

### Enum rcu\_osci\_type\_enum

Table 3-467. 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

### Enum rcu\_clock\_freq\_enum

Table 3-468. 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_I2C0	I2C0 clock

### rcu\_deinit

The description of rcu\_deinit is shown as below:

Table 3-469. 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	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset RCU */
rcu_deinit();
```

### rcu\_irc16m\_dfs\_to\_rf\_enable

The description of rcu\_irc16m\_dfs\_to\_rf\_enable is shown as below:

Table 3-470. Function rcu\_irc16m\_dfs\_to\_rf\_enable

Function name	rcu_irc16m_dfs_to_rf_enable
Function prototype	void rcu_irc16m_dfs_to_rf_enable(void);
Function descriptions	enable differential signal of IRC16M to RF module
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable differential signal of IRC16M to RF module */
rcu_irc16m_dfs_to_rf_enable();
```

### rcu\_irc16m\_dfs\_to\_rf\_disable

The description of rcu\_irc16m\_dfs\_to\_rf\_disable is shown as below:

Table 3-471. Function rcu\_irc16m\_dfs\_to\_rf\_disable

Function name	rcu_irc16m_dfs_to_rf_disable
Function prototype	void rcu_irc16m_dfs_to_rf_disable(void);
Function descriptions	disable differential signal of IRC16M to RF module
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable differential signal of IRC16M to RF module */
```

```
rcu_irc16m_dfs_to_rf_disable();
```

### rcu\_periph\_clock\_enable

The description of rcu\_periph\_clock\_enable is shown as below:

Table 3-472. 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 3-460. 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 3-473. Function rcu\_periph\_clock\_disable

Function name	rcu_periph_clock_disable
Function prototype	void rcu_periph_clock_disable(rcu_periph_enum periph);
Function descriptions	disable the peripherals clock
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to <a href="#">Table 3-460. Enum rcu_periph_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the USART0 clock */
rcu_periph_clock_disable(RCU_USART0);
```

### rcu\_fmc\_clock\_sleep\_enable

The description of rcu\_fmc\_clock\_sleep\_enable is shown as below:

Table 3-474. Function rcu\_periph\_clock\_sleep\_enable

Function name	rcu_fmc_clock_sleep_enable
Function prototype	void rcu_fmc_clock_sleep_enable(void);
Function descriptions	enable the FMC clock when in sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the FMC clock when in sleep mode */
rcu_fmc_clock_sleep_enable();
```

### rcu\_fmc\_clock\_sleep\_disable

The description of rcu\_fmc\_clock\_sleep\_disable is shown as below:

Table 3-475. Function rcu\_fmc\_clock\_sleep\_disable

Function name	rcu_fmc_clock_sleep_disable
Function prototype	void rcu_fmc_clock_sleep_disable(void);
Function descriptions	disable the FMC clock when in sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the FMC clock when in sleep mode */
```

```
rcu_fmc_clock_sleep_disable();
```

### rcu\_periph\_reset\_enable

The description of rcu\_periph\_reset\_enable is shown as below:

Table 3-476. 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 3-461. Enum rcu_periph_reset_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI reset */
```

```
rcu_periph_reset_enable(RCU_SPIRST);
```

### rcu\_periph\_reset\_disable

The description of rcu\_periph\_reset\_disable is shown as below:

Table 3-477. 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 3-461. Enum rcu_periph_reset_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI reset */
rcu_periph_reset_disable(RCU_SPIRST);
```

### rcu\_bkp\_reset\_enable

The description of rcu\_bkp\_reset\_enable is shown as below:

Table 3-478. 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 3-479. 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\_rfpll\_cal\_enable

The description of rcu\_rfpll\_cal\_enable is shown as below:

Table 3-480. 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 3-481. Function rcu\_rfppl\_cal\_disable

Function name	rcu_rfppl_cal_disable
Function prototype	void rcu_rfppl_cal_disable(void);
Function descriptions	disable the RF PLL calculation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the RF PLL calculation */
```

```
rcu_rfppl_cal_disable();
```

### rcu\_control\_unit\_powerup

The description of rcu\_control\_unit\_powerup is shown as below:

Table 3-482. 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
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 3-483. Function rcu\_control\_unit\_powerdown**

<b>Function name</b>	rcu_control_unit_powerdown
<b>Function prototype</b>	void rcu_control_unit_powerdown (rcu_unit_enum rcu_unit);
<b>Function descriptions</b>	power down the clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rcu_unit</b>	RCU unit
<i>RCU_UNIT_HXTAL</i>	power down the HXTAL
<i>RCU_UNIT_PLLDIG</i>	power down the PLLDIG
<i>RCU_UNIT_RFPLL</i>	disenable the RF PLL calculation
<i>RCU_UNIT_LDOANA</i>	power down LDO analog
<i>RCU_UNIT_LDOCLK</i>	power down the LDO clock
<i>RCU_UNIT_BANDGAP</i>	power down the BandGap
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* power down the HXTAL */
```

```
rcu_control_unit_powerdown(RCU_UNIT_HXTAL);
```

## rcu\_system\_clock\_source\_config

The description of rcu\_system\_clock\_source\_config is shown as below:

**Table 3-484. 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_IRC16M</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_PLLDIG as the CK_SYS source

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

Example:

```
/* configure the CK_HXTAL as the CK_SYS source */
rcu_system_clock_source_config(RCU_CKSYSSRC_HXTAL);
```

### rcu\_system\_clock\_source\_get

The description of rcu\_system\_clock\_source\_get is shown as below:

**Table 3-485. 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>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	RCU_SCSS_IRC16M/RCU_SCSS_HXTAL/RCU_SCSS_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 3-486. 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>	-
Input parameter{in}	

<b>ck_ahb</b>	AHB clock prescaler selection
<i>RCU_AHB_CKSYS_DIVx</i>	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 3-487. 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_DIV1</i>	select CK_AHB as CK_APB1
<i>RCU_APB1_CKAHB_DIV2</i>	select CK_AHB/2 as CK_APB1
<i>RCU_APB1_CKAHB_DIV4</i>	select CK_AHB/4 as CK_APB1
<i>RCU_APB1_CKAHB_DIV8</i>	select CK_AHB/8 as CK_APB1
<i>RCU_APB1_CKAHB_DIV16</i>	select CK_AHB/16 as CK_APB1
<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 3-488. 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_CK_AHB_D</i> <i>IV1</i>	select CK_AHB as CK_APB2
<i>RCU_APB2_CK_AHB_D</i> <i>IV2</i>	select CK_AHB/2 as CK_APB2
<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 3-489. Function rcu\_ckout0\_config**

<b>Function name</b>	rcu_ckout0_config
<b>Function prototype</b>	void rcu_ckout0_config(uint32_t ckout0_src, uint32_t ckout0_div);
<b>Function descriptions</b>	configure the CK_OUT0 clock source and divider
<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>	select IRC16M

<i>C16M</i>	
<i>RCU_CKOUT0SRC_LXTAL</i>	select LXTAL
<i>RCU_CKOUT0SRC_HXTAL</i>	select HXTAL
<i>RCU_CKOUT0SRC_PLLDIG</i>	select PLLDIG
<i>RCU_CKOUT0SRC_IRC32K</i>	select IRC32K
<i>RCU_CKOUT0SRC_KSYS</i>	select system clock
<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 3-490. Function rcu\_ckout1\_config**

<b>Function name</b>	rcu_ckout1_config
<b>Function prototype</b>	void rcu_ckout1_config(uint32_t ckout1_src, uint32_t ckout1_div);
<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_KSYS</i>	select system clock
<i>RCU_CKOUT1SRC_IRC16M</i>	select IRC16M
<i>RCU_CKOUT1SRC_HXTAL</i>	select HXTAL
<i>RCU_CKOUT1SRC_PLLDIG</i>	select PLLDIG

Input parameter{in}	
<b>ckout1_div</b>	CK_OUT1 divider
<i>RCU_CKOUT1_DIVx</i>	CK_OUT1 is divided by x(x=1,2,3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the HXTAL as CK_OUT1 clock source */
```

```
rcu_ckout1_config(RCU_CKOUT1SRC_HXTAL, RCU_CKOUT1_DIV1);
```

### rcu\_plldig\_config

The description of rcu\_plldig\_config is shown as below:

**Table 3-491. Function rcu\_pll\_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
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>plldig_clk</b>	PLLDIG output clock selection
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLLDIG output 192Mhz clock frequency */
```

```
rcu_plldig_config(RCU_PLLDIG_192M);
```

### rcu\_plldigdiv\_sys\_config

The description of rcu\_plldigdiv\_sys\_config is shown as below:

**Table 3-492. Function rcu\_plldigdiv\_sys\_config**

<b>Function name</b>	rcu_plldigdiv_sys_config
----------------------	--------------------------

<b>Function prototype</b>	void rcu_plldigdiv_sys_config(uint32_t plldigdiv_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>plldigdiv_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>	
-	-

Example:

```
/* configure PLLDIG clock divider 2 for system clock */
```

```
rcu_plldigdiv_sys_config (RCU_PLLDIG_SYS_DIV2);
```

### rcu\_rtc\_clock\_config

The description of rcu\_rtc\_clock\_config is shown as below:

**Table 3-493. 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_RTCSRC_NONE</i>	no clock selected
<i>RCU_RTCSRC_LXTAL</i>	select CK_LXTAL as RTC source clock
<i>RCU_RTCSRC_IRC32K</i>	select CK_IRC32K as RTC source clock
<i>RCU_RTCSRC_HXTAL_DIV_RTCDIV</i>	select CK_HXTAL/RTCDIV as RTC source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the RTC clock source selection */
```

```
rcu_rtc_clock_config(RCU_RTCSRC_IRC32K);
```

## rcu\_rtc\_div\_config

The description of rcu\_rtc\_div\_config is shown as below:

**Table 3-494. 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_DIVx</i>	RTCDIV clock select CK_HXTAL/x, x = 1....32
<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\_trng\_div\_config

The description of rcu\_trng\_div\_config is shown as below:

**Table 3-495. Function rcu\_trng\_div\_config**

<b>Function name</b>	rcu_trng_div_config
<b>Function prototype</b>	void rcu_trng_div_config(uint32_t trng_div);
<b>Function descriptions</b>	configure the frequency division of the TRNG source clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>trng_div</b>	TRNG clock frequency division
<i>RCU_TRNG_DIVx</i>	RTCDIV clock select CK_HXTAL/x, x = 1....32
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* TRNG clock divide 2 */
```

```
rcu_trng_div_config (RCU_TRNG_DIV2);
```

## rcu\_i2c0\_clock\_config

The description of rcu\_i2c0\_clock\_config is shown as below:

**Table 3-496. Function rcu\_i2c0\_clock\_config**

Function name	rcu_i2c0_clock_config
Function prototype	void rcu_i2c0_clock_config(uint32_t i2c0_clock_source);
Function descriptions	configure the I2C0 clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
i2c0_clock_source	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
Output parameter{out}	
-	-
Return value	
-	-

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 3-497. Function rcu\_usart0\_clock\_config**

Function name	rcu_usart0_clock_config
Function prototype	void rcu_usart0_clock_config(uint32_t usart0_clock_source);
Function descriptions	configure the USART0 clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
usart0_clock_source	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>KSYS</i>	
<i>RCU_USART0SRC_LXTAL</i>	CK_LXTAL selected as USART0 source clock
<i>RCU_USART0SRC_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\_irc16m\_div\_config

The description of rcu\_irc16m\_div\_config is shown as below:

**Table 3-498. 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 3-499. 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
<i>RCU_TIMER_PSC_MU</i> <i>L2</i>	If CK_APBx = CK_AHB or CK_APBx = CK_AHB/2, CK_TIMERx = CK_AHB, else CK_TIMERx = 2 x CK_APBx
<i>RCU_TIMER_PSC_MU</i> <i>L4</i>	If CK_APBx = CK_AHB or CK_APBx = CK_AHB/2 or CK_APBx = CK_AHB/4, CK_TIMERx = CK_AHB, else CK_TIMERx = 4 x CK_APBx
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER clock source */
rcu_timer_clock_prescaler_config(RCU_TIMER_PSC_MUL4);
```

### rcu\_flag\_get

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

**Table 3-500. 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>	
<b>flag</b>	the clock stabilization and peripheral reset flags, refer to <a href="#">Table 3-463. 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 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 3-501. 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>	
-	-
<b>Return value</b>	
-	-

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 3-502. 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 3-463. 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 3-503. 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 3-465. Enum rcu_int_flag_clear_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the interrupt HXTAL stabilization interrupt flag */
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

## rcu\_interrupt\_enable

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

**Table 3-504. Function rcu\_interrupt\_enable**

<b>Function name</b>	rcu_interrupt_enable
<b>Function prototype</b>	void rcu_interrupt_enable(rcu_int_enum interrupt);
<b>Function descriptions</b>	enable 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 3-466. Enum rcu_int_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the HXTAL stabilization interrupt */
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

## rcu\_interrupt\_disable

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

**Table 3-505. 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 3-466. 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);
```

## rcu\_lxtal\_drive\_capability\_config

The description of rcu\_lxtal\_drive\_capability\_config is shown as below:

**Table 3-506. 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_LOW ER_DRIVE</i>	lower driving capability
<i>RCU_LXTALDRI_HIGH _DRIVE</i>	high driving capability
<i>RCU_LXTALDRI_HIGH ER_DRIVE</i>	higher driving capability
<i>RCU_LXTALDRI_HIGH EST_DRIVEI</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_LXTALDRI_LOWER_DRIVE);
```

### rcu\_osci\_stab\_wait

The description of rcu\_osci\_stab\_wait is shown as below:

**Table 3-507. 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 3-467. 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 3-508. 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 3-467. 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 3-509. 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 3-467. 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 3-510. 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 3-467. 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>	
-	-

Return value	
-	-

Example:

```
/* enable the high speed crystal oscillator bypass mode */
```

```
rcu_osc_bypass_mode_enable(RCU_HXTAL);
```

### rcu\_osc\_bypass\_mode\_disable

The description of rcu\_osc\_bypass\_mode\_disable is shown as below:

**Table 3-511. Function rcu\_osc\_bypass\_mode\_disable**

Function name	rcu_osc_bypass_mode_disable
Function prototype	void rcu_osc_bypass_mode_disable(rcu_osc_type_enum osci);
Function descriptions	disable the oscillator bypass mode
Precondition	HXTALEN or LXTALEN must be reset before it
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to <a href="#">Table 3-467. Enum rcu_osc_type_enum</a>
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the high speed crystal oscillator bypass mode */
```

```
rcu_osc_bypass_mode_disable(RCU_HXTAL);
```

### rcu\_rf\_hxtal\_clock\_monitor\_enable

The description of rcu\_rf\_hxtal\_clock\_monitor\_enable is shown as below:

**Table 3-512. Function rcu\_rf\_hxtal\_clock\_monitor\_enable**

Function name	rcu_rf_hxtal_clock_monitor_enable
Function prototype	void rcu_rf_hxtal_clock_monitor_enable(void);
Function descriptions	enable the RF HXTAL clock monitor
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

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 3-513. 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 3-514. 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>	
-	-

Return value	
-	-

Example:

```
/* set the IRC16M adjust value */
```

```
rcu_irc16m_adjust_value_set(0x10);
```

### rcu\_voltage\_key\_unlock

The description of rcu\_voltage\_key\_unlock is shown as below:

**Table 3-515. Function rcu\_voltage\_key\_unlock**

Function name	rcu_voltage_key_unlock
Function prototype	void rcu_voltage_key_unlock(void);
Function descriptions	unlock the voltage key
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

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 3-516. Function rcu\_deepsleep\_voltage\_set**

Function name	rcu_deepsleep_voltage_set
Function prototype	void rcu_deepsleep_voltage_set(uint32_t dsvol);
Function descriptions	set the deep-sleep mode voltage value
Precondition	-
The called functions	-
Input parameter{in}	
<b>dsvol</b>	deep sleep mode voltage
<i>RCU_DEEPSLEEP_V_1_1</i>	the core voltage is 1.1V in deep-sleep mode
<i>RCU_DEEPSLEEP_V_1_0</i>	the core voltage is 1.0V in deep-sleep mode

1_0	
RCU_DEEPSLEEP_V_0_9	the core voltage is 0.9V in deep-sleep mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the deep-sleep mode voltage */
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1_0);
```

### rcu\_clock\_freq\_get

The description of rcu\_clock\_freq\_get is shown as below:

**Table 3-517. 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>	-
Input parameter{in}	
<b>clock</b>	Clock frequency, refers to <a href="#">Table 3-468. Enum rcu_clock_freq_enum</a>
Output parameter{out}	
-	-
Return value	
<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);
```

## 3.19. 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 [3.19.1](#), the RTC firmware functions are introduced in chapter [3.19.2](#).

### 3.19.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

**Table 3-518. RTC Registers**

Registers	Descriptions
RTC_TIME	time of day register
RTC_DATE	date register
RTC_CTL	control register
RTC_STAT	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
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_BKPx(x = 0, 1, 2, ..., 18, 19)	backup register

### 3.19.2. Descriptions of Peripheral functions

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

**Table 3-519. 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

Function name	Function description
rtc_alarm_config	configure RTC alarm
rtc_alarm_subsecond_config	configure subsecond of RTC alarm
rtc_alarm_get	get RTC alarm
rtc_alarm_subsecond_get	get RTC alarm subsecond
rtc_alarm_enable	enable RTC alarm
rtc_alarm_disable	disable RTC alarm
rtc_timestamp_enable	enable RTC time-stamp
rtc_timestamp_disable	disable RTC time-stamp
rtc_timestamp_get	get RTC timestamp time and date
rtc_timestamp_subsecond_get	get RTC time-stamp subsecond
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
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_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

## Structure rtc\_parameter\_struct

**Table 3-520. 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 3-521. 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 3-522. 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 3-523. 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 3-524. 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>	-
<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;
```

```
error_status = rtc_deinit();
```

### rtc\_init

The description of rtc\_init is shown as below:

**Table 3-525. 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 3-520. 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 3-526. 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>	
-	-

Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/*enter RTC init mode*/
```

```
ErrStatus error_status;
```

```
error_status = rtc_init_mode_enter ();
```

### rtc\_init\_mode\_exit

The description of rtc\_init\_mode\_exit is shown as below:

**Table 3-527. Function rtc\_init\_mode\_exit**

Function name	rtc_init_mode_exit
Function prototype	void rtc_init_mode_exit(void);
Function descriptions	exit RTC init mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*exit RTC init mode*/
```

```
rtc_init_mode_exit ();
```

### rtc\_register\_sync\_wait

The description of rtc\_register\_sync\_wait is shown as below:

**Table 3-528. Function rtc\_register\_sync\_wait**

Function name	rtc_register_sync_wait
Function prototype	ErrStatus rtc_register_sync_wait(void);
Function descriptions	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
<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;
```

```
error_status = rtc_register_sync_wait ();
```

### rtc\_current\_time\_get

The description of rtc\_current\_time\_get is shown as below:

**Table 3-529. 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 3-520. 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 3-530. 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>	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
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 3-531. Function rtc\_alarm\_config**

Function name	rtc_alarm_config
Function prototype	void rtc_alarm_config(uint8_t rtc_alarm, rtc_alarm_struct* rtc_alarm_time)
Function descriptions	configure RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm
RTC_ALARM0	alarm 0
RTC_ALARM1	alarm 1
Input parameter{in}	
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 3-521. Structure rtc_alarm_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

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 3-532. 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> 4	mask RTC_ALRM0SS_SSC[14:10], and RTC_ALRM0SS_SSC[9:0] is to be compared
<i>RTC_MASKSSC_11_14</i> 4	mask RTC_ALRM0SS_SSC[14:11], and RTC_ALRM0SS_SSC[10:0] is to be compared
<i>RTC_MASKSSC_12_14</i> 4	mask RTC_ALRM0SS_SSC[14:12], and RTC_ALRM0SS_SSC[11:0] is to be compared
<i>RTC_MASKSSC_13_14</i> 4	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_NON</i>	mask none, and RTC_ALRM0SS_SSC[14:0] is to be compared

<i>E</i>	
<b>Input parameter{in}</b>	
<b>subsecond</b>	alarm subsecond value(0x000 - 0x7FFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-533. Function rtc\_alarm\_get**

<b>Function name</b>	rtc_alarm_get
<b>Function prototype</b>	void rtc_alarm_get(rtc_alarm_struct* rtc_alarm_time);
<b>Function descriptions</b>	get 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>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 3-521. Structure rtc_alarm_struct</a>
<b>Return value</b>	
-	-

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 3-534. Function rtc\_alarm\_subsecond\_get**

<b>Function name</b>	rtc_alarm_subsecond_get
----------------------	-------------------------

<b>Function prototype</b>	uint32_t rtc_alarm_subsecond_get(uint8_t rtc_alarm)
<b>Function descriptions</b>	get RTC alarm subsecond
<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>	
<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 3-535. 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 3-536. 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>	-
<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>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/*disable RTC alarm0*/
```

```
ErrStatus error_status;
```

```
error_status = rtc_alarm_disable(RTC_ALARM0);
```

### rtc\_timestamp\_enable

The description of rtc\_timestamp\_enable is shown as below:

Table 3-537. 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>	-
<b>Input parameter{in}</b>	
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-538. 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 3-539. 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 3-523. 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 3-540. Function `rtc_timestamp_subsecond_get`**

<b>Function name</b>	<code>rtc_timestamp_subsecond_get</code>
<b>Function prototype</b>	<code>uint32_t rtc_timestamp_subsecond_get(void);</code>
<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>	
<code>uint32_t</code>	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 3-541. Function `rtc_tamper_enable`**

<b>Function name</b>	<code>rtc_tamper_enable</code>
<b>Function prototype</b>	<code>void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);</code>
<b>Function descriptions</b>	enable RTC tamper
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>rtc_tamper</code>	pointer to a <code>rtc_tamper_struct</code> structure which contains parameters for RTC tamper configuration, the structure members can refer to members of the structure <a href="#">Table 3-523. Structure <code>rtc_tamper_struct</code></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 3-542. 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 3-543. 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 3-544. Function rtc\_tamper\_without\_bkp\_seset**

Function name	rtc_tamper_without_bkp_seset
Function prototype	void rtc_tamper_without_bkp_seset(uint32_t ne_source)
Function descriptions	tamperx event does not erase the RTC_BKP registers
Precondition	-
The called functions	-
Input parameter{in}	
ne_source	specify which tamper source will not trigger the RTC_BKP registers reset
RTC_TAMPXNOER_NONE	both tamper0 and tamper1 event will trigger RTC_BKP registers reset
RTC_TAMPXNOER_TP0	tamper0 event will not trigger RTC_BKP registers reset
RTC_TAMPXNOER_TP1	tamper1 event will not trigger RTC_BKP registers reset
RTC_TAMPXNOER_TP0_TP1	neither tamper0 nor tamper1 event will trigger RTC_BKP registers reset
Output parameter{out}	
-	-
Return value	
-	-

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 3-545. Function rtc\_output\_pin\_select**

Function name	rtc_output_pin_select
Function prototype	void rtc_output_pin_select(uint32_t pin);
Function descriptions	select the RTC output pin
Precondition	-
The called functions	-

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

Example:

```
/* select the rtc output pad is PC15 */
```

```
rtc_output_pad_select(RTC_OUT_PC15);
```

### rtc\_alarm\_output\_config

The description of rtc\_alarm\_output\_config is shown as below:

**Table 3-546. Function rtc\_alter\_output\_config**

<b>Function name</b>	rtc_alarm_output_config
<b>Function prototype</b>	void rtc_alarm_output_config(uint32_t source, uint32_t mode);
<b>Function descriptions</b>	configure rtc alarm output source
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<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
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure rtc alternate output source */
```

```
rtc_alter_output_config(RTC_ALARM0_LOW, RTC_ALARM_OUTPUT_PP);
```

### rtc\_calibration\_output\_config

The description of rtc\_calibration\_output\_config is shown as below:

**Table 3-547. Function rtc\_calibration\_output\_config**

<b>Function name</b>	rtc_calibration_output_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_5</i> <i>12HZ</i>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 512Hz signal
<i>RTC_CALIBRATION_1</i> <i>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 3-548. 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

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* adjust the daylight saving time by adding one hour from the current time */
```

```
rtc_hour_adjust(RTC_CTL_A1H);
```

### rtc\_second\_adjust

The description of rtc\_second\_adjust is shown as below:

**Table 3-549. Function rtc\_second\_adjust**

Function name	rtc_second_adjust
Function prototype	ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);
Function descriptions	adjust RTC second or subsecond value of current time
Precondition	-
The called functions	-
Input parameter{in}	
<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
Input parameter{in}	
<b>minus</b>	number of subsecond to minus from current time(0x0 - 0x7FFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* adjust RTC second or subsecond value of current time */
```

```
ErrStatus error_status;
```

```
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 3-550. Function rtc\_bypass\_shadow\_enable**

Function name	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>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-551. Function rtc\_bypass\_shadow\_disable**

<b>Function name</b>	rtc_bypass_shadow_disable
<b>Function prototype</b>	void rtc_bypass_shadow_disable (void);
<b>Function descriptions</b>	disable RTC bypass shadow registers 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 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 3-552. Function rtc\_refclock\_detection\_enable**

<b>Function name</b>	rtc_refclock_detection_enable
<b>Function prototype</b>	ErrStatus rtc_refclock_detection_enable(void);

<b>Function descriptions</b>	enable RTC reference clock detection 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:

```
/* enable RTC reference clock detection function*/
```

```
ErrStatus error_status;
```

```
error_status = rtc_refclock_detection_enable();
```

### rtc\_refclock\_detection\_disable

The description of rtc\_refclock\_detection\_disable shown as below:

**Table 3-553. Function rtc\_refclock\_detection\_disable**

<b>Function name</b>	rtc_refclock_detection_disable
<b>Function prototype</b>	ErrStatus rtc_refclock_detection_disable(void);
<b>Function descriptions</b>	disable RTC reference clock detection 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 reference clock detection function*/
```

```
ErrStatus error_status;
```

```
error_status = rtc_refclock_detection_disable ();
```

### rtc\_wakeup\_enable

The description of rtc\_refclock\_detection\_disable is shown as below:

**Table 3-554. Function rtc\_wakeup\_enable**

<b>Function name</b>	rtc_wakeup_enable
----------------------	-------------------

<b>Function prototype</b>	void rtc_wakeup_enable(void);
<b>Function descriptions</b>	enable 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>	
-	-

Example:

```
/* enable RTC auto wakeup function */
rtc_wakeup_enable();
```

### rtc\_wakeup\_disable

The description of rtc\_wakeup\_disable is shown as below:

**Table 3-555. 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;
error_status = rtc_wakeup_disable ();
```

### rtc\_wakeup\_clock\_set

The description of rtc\_wakeup\_clock\_set is shown as below:

**Table 3-556. 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_DIV16	RTC auto wakeup timer clock is RTC clock divided by 16
WAKEUP_RTCK_DIV8	RTC auto wakeup timer clock is RTC clock divided by 8
WAKEUP_RTCK_DIV4	RTC auto wakeup timer clock is RTC clock divided by 4
WAKEUP_RTCK_DIV2	RTC auto wakeup timer clock is RTC clock divided by 2
WAKEUP_CKSPRE	RTC auto wakeup timer clock is ckspre
WAKEUP_CKSPRE_2EXP16	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;
```

```
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 3-557. 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
uint16_t	0x0000-0xffff
<b>Output parameter{out}</b>	
-	-

Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* set wakeup timer value */
```

```
ErrStatus error_status;
```

```
error_status = rtc_wakeup_timer_set(0XFFEE);
```

### rtc\_wakeup\_timer\_get

The description of rtc\_wakeup\_timer\_set is shown as below:

**Table 3-558. Function rtc\_wakeup\_timer\_get**

Function name	rtc_wakeup_timer_get
Function prototype	uint16_t rtc_wakeup_timer_get(void);
Function descriptions	set wakeup timer value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	0-0xFFFF

Example:

```
/* get wakeup timer value */
```

```
uint32_t wakeup_time;
```

```
wakeup_time = rtc_wakeup_timer();
```

### rtc\_smooth\_calibration\_config

The description of rtc\_smooth\_calibration\_config is shown as below:

**Table 3-559. Function rtc\_smooth\_calibration\_config**

Function name	rtc_smooth_calibration_config
Function prototype	ErrStatus rtc_smooth_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);
Function descriptions	configure RTC smooth calibration
Precondition	-
The called functions	-
Input parameter{in}	
window	select calibration window

<i>RTC_CALIBRATION_WINDOW_32S</i>	2exp20 RTCCLK cycles, 32s if RTCCLK = 32768 Hz
<i>RTC_CALIBRATION_WINDOW_16S</i>	2exp19 RTCCLK cycles, 16s if RTCCLK = 32768 Hz
<i>RTC_CALIBRATION_WINDOW_8S</i>	2exp18 RTCCLK cycles, 8s if RTCCLK = 32768 Hz
<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 - 0xFF)
<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 3-560. Function `rtc_coarse_calibration_enable`**

<b>Function name</b>	<code>rtc_coarse_calibration_enable</code>
<b>Function prototype</b>	<code>ErrStatus rtc_coarse_calibration_enable(void);</code>
<b>Function descriptions</b>	enable RTC coarse calibration
<b>Precondition</b>	-
<b>The called functions</b>	<code>rtc_init_mode_enter/rtc_init_mode_exit</code>
<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 3-561. Function 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>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* rtc_coarse_calibration_disable */
```

```
ErrStatus error_status;
```

```
error_status = rtc_coarse_calibration_disable ();
```

### rtc\_coarse\_calibration\_config

The description of rtc\_coarse\_calibration\_config is shown as below:

**Table 3-562. Function rtc\_coarse\_calibration\_config**

<b>Function name</b>	rtc_coarse_calibration_config
<b>Function prototype</b>	ErrStatus rtc_coarse_calibration_config(uint8_t direction, uint8_t step);
<b>Function descriptions</b>	config coarse calibration direction and step
<b>Precondition</b>	-
<b>The called functions</b>	rtc_init_mode_enter/rtc_init_mode_exit
<b>Input parameter{in}</b>	
<b>direction</b>	coarse calibration direction
<i>CALIB_INCREASE</i>	Increase calendar update frequency
<i>CALIB_DECREASE</i>	Decrease calendar update frequency
<b>Input parameter{in}</b>	
<b>step</b>	coarse calibration step
<i>0x00-0x1F</i>	COSD=0: 0x00: +0PPM

	0x01: +4PPM(approximate value) 0x02: +8PPM (approximate value) ..... 0x1F: +126PPM (approximate value) COSD=1: 0x00: -0PPM 0x01: -2PPM(approximate value) 0x02: -4PPM (approximate value) ..... 0x1F: -63PPM (approximate value)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* config coarse calibration direction and step */
```

```
ErrStatus error_status;
```

```
error_status = rtc_coarse_calibration_config (INCREASE, 0x01);
```

### rtc\_flag\_get

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

**Table 3-563. 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 flag_status;
```

```
flag_status = rtc_flag_get(RTC_FLAG_TS);
```

### rtc\_flag\_clear

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

**Table 3-564. Function rtc\_flag\_clear**

<b>Function name</b>	rtc_flag_clear
<b>Function prototype</b>	void rtc_flag_clear(uint32_t flag);
<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 3-565. 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 3-566. 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);
```

## 3.20. SPI

The SPI module can communicate with external devices using the SPI protocol. The SPI registers are listed in chapter [3.20.1](#), the SPI firmware functions are introduced in chapter [3.20.2](#).

### 3.20.1. Descriptions of Peripheral registers

SPI registers are listed in the table shown as below:

**Table 3-567. SPI 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

### 3.20.2. Descriptions of Peripheral functions

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

**Table 3-568. SPI firmware function**

Function name	Function description
spi_deinit	reset SPI 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
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

Function name	Function description
spi_data_frame_format_config	configure SPI data frame format
spi_data_transmit	SPI transmit data
spi_data_receive	SPI receive data
spi_bidirectional_transfer_config	configure SPI bidirectional transfer direction
spi_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_interrupt_enable	enable SPI interrupt
spi_interrupt_disable	disable SPI interrupt
spi_interrupt_flag_get	get SPI interrupt status
spi_flag_get	get SPI flag status

### Structure spi\_parameter\_struct

**Table 3-569. 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_xBIT, x=8/16)
nss	SPI NSS control by hardware or software (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	SPI big endian or little endian (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	SPI clock phase and polarity (SPI_CK_PL_LOW_PH_1EDGE, SPI_CK_PL_HIGH_PH_1EDGE, SPI_CK_PL_LOW_PH_2EDGE, SPI_CK_PL_HIGH_PH_2EDGE)
prescale	SPI prescale factor (SPI_PSC_n (n=2,4,8,16,32,64,128,256))

## spi\_deinit

The description of spi\_deinit is shown as below:

**Table 3-570. Function spi\_deinit**

<b>Function name</b>	spi_deinit
<b>Function prototype</b>	void spi_deinit(void);
<b>Function descriptions</b>	reset SPI 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 SPI */
spi_deinit();
```

## spi\_struct\_para\_init

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

**Table 3-571. 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>	SPI init parameter struct, the structure members can refer to <a href="#">Table 3-569. Structure spi_parameter_struct</a>
<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 3-572. Function spi\_init**

<b>Function name</b>	spi_init
<b>Function prototype</b>	void spi_init(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_struct</b>	SPI parameter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-569. Structure spi_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize SPI */

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_PSC_8;
spi_init_struct.endian          = SPI_ENDIAN_MSB;

spi_init(SPI, &spi_init_struct);
```

## spi\_enable

The description of spi\_enable is shown as below:

**Table 3-573. Function spi\_enable**

<b>Function name</b>	spi_enable
<b>Function prototype</b>	void spi_enable(void);
<b>Function descriptions</b>	enable SPI
<b>Precondition</b>	-
<b>The called functions</b>	-

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

Example:

```
/* enable SPI */
```

```
spi_enable();
```

### spi\_disable

The description of spi\_disable is shown as below:

**Table 3-574. Function spi\_disable**

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

Example:

```
/* disable SPI */
```

```
spi_disable();
```

### spi\_nss\_output\_enable

The description of spi\_nss\_output\_enable is shown as below:

**Table 3-575. Function spi\_nss\_output\_enable**

Function name	spi_nss_output_enable
Function prototype	void spi_nss_output_enable(void);
Function descriptions	enable SPI NSS output function
Precondition	-
The called functions	-
Input parameter{in}	

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

Example:

```
/* enable SPI NSS output */
spi_nss_output_enable();
```

### spi\_nss\_output\_disable

The description of spi\_nss\_output\_disable is shown as below:

**Table 3-576. Function spi\_nss\_output\_disable**

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

Example:

```
/* disable SPI NSS output */
spi_nss_output_disable();
```

### spi\_nss\_internal\_high

The description of spi\_nss\_internal\_high is shown as below:

**Table 3-577. Function spi\_nss\_internal\_high**

Function name	spi_nss_internal_high
Function prototype	void spi_nss_internal_high(void);
Function descriptions	SPI NSS pin high level in software mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI NSS pin is pulled high level in software mode */
```

```
spi_nss_internal_high();
```

### spi\_nss\_internal\_low

The description of spi\_nss\_internal\_low is shown as below:

**Table 3-578. Function spi\_nss\_internal\_low**

Function name	spi_nss_internal_low
Function prototype	void spi_nss_internal_low(void);
Function descriptions	SPI NSS pin low level in software mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low();
```

### spi\_dma\_enable

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

**Table 3-579. Function spi\_dma\_enable**

Function name	spi_dma_enable
Function prototype	void spi_dma_enable(uint8_t dma);
Function descriptions	enable SPI DMA function
Precondition	-
The called functions	-
Input parameter{in}	
dma	SPI DMA mode
SPI_DMA_TRANSMIT	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 SPI transmit data DMA function */
spi_dma_enable(SPI_DMA_TRANSMIT);
```

### spi\_dma\_disable

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

**Table 3-580. Function spi\_dma\_disable**

<b>Function name</b>	spi_dma_disable
<b>Function prototype</b>	void spi_dma_disable(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>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 SPI transmit data DMA function */
spi_dma_disable(SPI_DMA_TRANSMIT);
```

### spi\_data\_frame\_format\_config

The description of spi\_data\_frame\_format\_config is shown as below:

**Table 3-581. Function spi\_data\_frame\_format\_config**

<b>Function name</b>	spi_data_frame_format_config
<b>Function prototype</b>	void spi_data_frame_format_config(uint16_t frame_format);
<b>Function descriptions</b>	configure SPI data frame format
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>frame_format</b>	SPI frame size
<i>SPI_FRAME_SIZE_xBIT</i>	SPI frame size is x bits,x=8/16
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SPI data frame format size is 16 bits */
```

```
spl_data_frame_format_config(SPI_FRAME_SIZE_16BIT);
```

### spl\_data\_transmit

The description of spl\_data\_transmit is shown as below:

**Table 3-582. Function spl\_data\_transmit**

<b>Function name</b>	spl_data_transmit
<b>Function prototype</b>	void spl_data_transmit(uint16_t data);
<b>Function descriptions</b>	SPI transmit data
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>data</b>	16-bit data
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI transmit data */
```

```
spl_data_transmit(spl_send_array[send_n]);
```

### spl\_data\_receive

The description of spl\_data\_receive is shown as below:

**Table 3-583. Function spl\_data\_receive**

<b>Function name</b>	spl_data_receive
<b>Function prototype</b>	uint16_t spl_data_receive(void);
<b>Function descriptions</b>	SPI receive data
<b>Precondition</b>	-
<b>The called functions</b>	-

Output parameter{out}	
-	-
Return value	
uint16_t	16-bit data

Example:

```
/* SPI receive data */
```

```
spi_receive_array[receive_n] = spi_data_receive();
```

### spi\_bidirectional\_transfer\_config

The description of spi\_bidirectional\_transfer\_config is shown as below:

**Table 3-584. Function spi\_bidirectional\_transfer\_config**

Function name	spi_bidirectional_transfer_config
Function prototype	void spi_bidirectional_transfer_config(uint32_t transfer_direction);
Function descriptions	configure SPI bidirectional transfer direction
Precondition	-
The called functions	-
Input parameter{in}	
transfer_direction	SPI transfer direction
SPI_BIDIRECTIONAL_TRANSMIT	SPI work in transmit-only mode
SPI_BIDIRECTIONAL_RECEIVE	SPI work in receive-only mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI_BIDIRECTIONAL_TRANSMIT);
```

### spi\_format\_error\_clear

The description of spi\_format\_error\_clear is shown as below:

**Table 3-585. Function spi\_format\_error\_clear**

Function name	spi_format_error_clear
Function prototype	void spi_format_error_clear(void);
Function descriptions	clear SPI format error flag status
Precondition	-

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

Example:

```
/* clear SPI format error flag status */
```

```
spi_format_error_clear();
```

### spi\_crc\_polynomial\_set

The description of spi\_crc\_polynomial\_set is shown as below:

**Table 3-586. Function spi\_crc\_polynomial\_set**

Function name	spi_crc_polynomial_set
Function prototype	void spi_crc_polynomial_set(uint16_t crc_poly);
Function descriptions	set SPI CRC polynomial
Precondition	-
The called functions	-
Input parameter{in}	
crc_poly	CRC polynomial value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set SPI CRC polynomial */
```

```
spi_crc_polynomial_set(CRC_VALUE);
```

### spi\_crc\_polynomial\_get

The description of spi\_crc\_polynomial\_get is shown as below:

**Table 3-587. Function spi\_crc\_polynomial\_get**

Function name	spi_crc_polynomial_get
Function prototype	uint16_t spi_crc_polynomial_get(void);
Function descriptions	get SPI CRC polynomial
Precondition	-
The called functions	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	16 bit CRC polynomial value

Example:

```
/* get SPI CRC polynomial */

uint16_t crc_val;

crc_val = spi_crc_polynomial_get();
```

### spi\_crc\_on

The description of spi\_crc\_on is shown as below:

**Table 3-588. Function spi\_crc\_on**

<b>Function name</b>	spi_crc_on
<b>Function prototype</b>	void spi_crc_on(void);
<b>Function descriptions</b>	turn on CRC function
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn on SPI CRC function */

spi_crc_on();
```

### spi\_crc\_off

The description of spi\_crc\_off is shown as below:

**Table 3-589. Function spi\_crc\_off**

<b>Function name</b>	spi_crc_off
<b>Function prototype</b>	void spi_crc_off(void);
<b>Function descriptions</b>	turn off CRC function
<b>Precondition</b>	-
<b>The called functions</b>	-

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

Example:

```
/* turn off SPI CRC function */
```

```
spi_crc_off();
```

### spi\_crc\_next

The description of spi\_crc\_next is shown as below:

**Table 3-590. Function spi\_crc\_next**

Function name	spi_crc_next
Function prototype	void spi_crc_next(void);
Function descriptions	SPI next data is CRC value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI next data is CRC value */
```

```
spi_crc_next();
```

### spi\_crc\_get

The description of spi\_crc\_get is shown as below:

**Table 3-591. Function spi\_crc\_get**

Function name	spi_crc_get
Function prototype	uint16_t spi_crc_get(uint8_t crc);
Function descriptions	get SPI CRC send value or receive value
Precondition	-
The called functions	-
Input parameter{in}	

<i>crc</i>	SPI crc value
<i>SPI_CRC_TX</i>	get transmit crc value
<i>SPI_CRC_RX</i>	get receive crc value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	16-bit CRC value

Example:

```
/* get SPI CRC send value */
```

```
uint16_t crc_val;
```

```
crc_val = spi_crc_get(SPI_CRC_TX);
```

### spi\_crc\_error\_clear

The description of spi\_crc\_error\_clear is shown as below:

**Table 3-592. Function spi\_crc\_error\_clear**

<b>Function name</b>	spi_crc_error_clear
<b>Function prototype</b>	void spi_crc_error_clear(void);
<b>Function descriptions</b>	clear SPI CRC error flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear SPI CRC error flag status */
```

```
spi_crc_error_clear();
```

### spi\_ti\_mode\_enable

The description of spi\_ti\_mode\_enable is shown as below:

**Table 3-593. Function spi\_ti\_mode\_enable**

<b>Function name</b>	spi_ti_mode_enable
<b>Function prototype</b>	void spi_ti_mode_enable(void);
<b>Function descriptions</b>	enable SPI TI mode
<b>Precondition</b>	-

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

Example:

```
/* enable SPI TI mode */
spi_ti_mode_enable();
```

### spi\_ti\_mode\_disable

The description of spi\_ti\_mode\_disable is shown as below:

**Table 3-594. Function spi\_ti\_mode\_disable**

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

Example:

```
/* disable SPI TI mode */
spi_ti_mode_disable();
```

### spi\_interrupt\_enable

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

**Table 3-595. Function spi\_interrupt\_enable**

Function name	spi_interrupt_enable
Function prototype	void spi_interrupt_enable(uint8_t interrupt);
Function descriptions	enable SPI interrupt
Precondition	-
The called functions	-

Input parameter{in}	
<b>interrupt</b>	SPI interrupt
<i>SPI_INT_TBE</i>	transmit buffer empty interrupt
<i>SPI_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI transmit buffer empty interrupt */
```

```
spi_interrupt_enable(SPI_INT_TBE);
```

### spi\_interrupt\_disable

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

**Table 3-596. Function spi\_interrupt\_disable**

<b>Function name</b>	spi_interrupt_disable
<b>Function prototype</b>	void spi_interrupt_disable(uint8_t interrupt);
<b>Function descriptions</b>	disable SPI interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>interrupt</b>	SPI interrupt
<i>SPI_INT_TBE</i>	transmit buffer empty interrupt
<i>SPI_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI transmit buffer empty interrupt */
```

```
spi_interrupt_disable(SPI_INT_TBE);
```

### spi\_interrupt\_flag\_get

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

Table 3-597. Function spi\_interrupt\_flag\_get

Function name	spi_interrupt_flag_get
Function prototype	FlagStatus spi_interrupt_flag_get(uint8_t interrupt);
Function descriptions	get SPI interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	SPI interrupt
SPI_INT_FLAG_TBE	transmit buffer empty interrupt
SPI_INT_FLAG_RBNE	receive buffer not empty interrupt
SPI_INT_FLAG_RXORERR	overrun interrupt
SPI_INT_FLAG_CONFERR	config error interrupt
SPI_INT_FLAG_CRCERR	CRC error interrupt
SPI_INT_FLAG_FERR	format error interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```

/* get SPI transmit buffer empty interrupt status */
If(RESET != spi_interrupt_flag_get(SPI_INT_FLAG_TBE)){
    while(RESET == spi_flag_get(SPI_FLAG_TBE));
    spi_data_transmit(spi_send_array[send_n++]);
}

```

## spi\_flag\_get

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

Table 3-598. Function spi\_flag\_get

Function name	spi_flag_get
Function prototype	FlagStatus spi_flag_get(uint8_t interrupt);
Function descriptions	get SPI flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	SPI flag status

<i>SPI_FLAG_TBE</i>	transmit buffer empty flag
<i>SPI_FLAG_RBNE</i>	receive buffer not empty flag
<i>SPI_FLAG_TRANS</i>	transmit on-going flag
<i>SPI_FLAG_RXORERR</i>	receive overrun error flag
<i>SPI_FLAG_CONFERR</i>	mode config error flag
<i>SPI_FLAG_CRCERR</i>	CRC error flag
<i>SPI_FLAG_FERR</i>	SPI format error interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```

/* get SPI transmit buffer empty flag status */
while(RESET == spi_flag_get(SPI_FLAG_TBE));

spi_data_transmit(spi_send_array[send_n++]);

```

## 3.21. SYSCFG

The SYSCFG registers are listed in chapter [3.21.1](#), the SYSCFG firmware functions are introduced in chapter [3.21.2](#).

### 3.21.1. Descriptions of Peripheral registers

SYSCFG registers are listed in the table shown as below:

**Table 3-599. SYSCFG Registers**

Registers	Descriptions
SYSCFG_CFG0	system 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_CPSCTL	I/O compensation control register
SYSCFG_CFG1	system configuration register 1
SYSCFG_SCFG	SYSCFG shared SRAM configuration register
SYSCFG_TIMERxCFG	TIMER trigger selection register(x = 0..2)

### 3.21.2. Descriptions of Peripheral functions

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

Table 3-600. 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_io_compensation_enable	enable I/O compensation cell
syscfg_io_compensation_disable	disable I/O compensation cell
syscfg_lock_config	connect TIMER0/15/16 break input to the selected parameter
syscfg_io_compensation_ready_flag_get	get the compensation ready flag
syscfg_sram_ownership_config	configure the ownership of the shared 32k SRAM
syscfg_boot_mode_get	get code start mode

### Enum syscfg\_code\_start\_enum

Table 3-601. Enum syscfg\_code\_start\_enum

enum name	Function description
FLASH_START	flash start mode
SRAM_START	sram start mode
SYSTEM_START	system start mode

### syscfg\_deinit

The description of syscfg\_deinit is shown as below:

Table 3-602. Function syscfg\_deinit

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

Example:

```
/* reset SYSCFG registers */
syscfg_deinit();
```

### syscfg\_exti\_line\_config

The description of syscfg\_exti\_line\_config is shown as below:

Table 3-603. Function syscfg\_exti\_line\_config

Function name	syscfg_exti_line_config
Function prototype	void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);
Function descriptions	configure the GPIO pin as EXTI Line
Precondition	-
The called functions	-
Input parameter{in}	
exti_port	specify the GPIO port used in EXTI
EXTI_SOURCE_GPIOx	x=A,B,C
exti_pin	specify the EXTI line
EXTI_SOURCE_PINx	x=0..15(GPIOA), x=0..4,11,12,13,15(GPIOB) , x=8,13,14,15 (GPIOC)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the GPIO pin as EXTI Line */
syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN0);
```

### syscfg\_io\_compensation\_enable

The description of syscfg\_io\_compensation\_enable is shown as below:

Table 3-604. Function syscfg\_io\_compensation\_enable

Function name	syscfg_io_compensation_enable
Function prototype	void syscfg_io_compensation_enable(void);
Function descriptions	enable I/O compensation cell
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I/O compensation cell */
syscfg_io_compensation_enable();
```

## syscfg\_io\_compensation\_disable

The description of syscfg\_io\_compensation\_disable is shown as below:

**Table 3-605. Function syscfg\_io\_compensation\_disable**

<b>Function name</b>	syscfg_io_compensation_disable
<b>Function prototype</b>	void syscfg_io_compensation_disable(void);
<b>Function descriptions</b>	disable I/O 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 I/O compensation cell */
syscfg_io_compensation_disable();
```

## syscfg\_lock\_config

The description of syscfg\_lock\_config is shown as below:

**Table 3-606. Function syscfg\_lock\_config**

<b>Function name</b>	syscfg_lock_config
<b>Function prototype</b>	void syscfg_lock_config(void);
<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>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* connect TIMER0/15/16 break input to the selected parameter */
syscfg_lock_config();
```

## syscfg\_io\_compensation\_ready\_flag\_get

The description of syscfg\_io\_compensation\_ready\_flag\_get is shown as below:

**Table 3-607. Function syscfg\_io\_compensation\_ready\_flag\_get**

<b>Function name</b>	syscfg_io_compensation_ready_flag_get
<b>Function prototype</b>	FlagStatus syscfg_io_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 / SET

Example:

```
/* get the compensation ready flag */
while(RESET != syscfg_io_compensation_ready_flag_get()){
}
```

## syscfg\_sram\_ownership\_config

The description of syscfg\_sram\_ownership\_config is shown as below:

**Table 3-608. Function syscfg\_sram\_ownership\_config**

<b>Function name</b>	syscfg_sram_ownership_config
<b>Function prototype</b>	void syscfg_sram_ownership_config(uint32_t sram_owner);
<b>Function descriptions</b>	configure the ownership of the shared 32k SRAM
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sram_owner</b>	the ownership of the shared SRAM
SRAMEN_WIRELESS	wireless
SRAMEN_CORE	core
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the ownership */
```

```
syscfg_sram_ownership_config(SRAMEN_WIRELESS);
```

## syscfg\_boot\_mode\_get

The description of syscfg\_boot\_mode\_get is shown as below:

**Table 3-609. Function syscfg\_boot\_mode\_get**

<b>Function name</b>	syscfg_boot_mode_get
<b>Function prototype</b>	syscfg_code_start_enum syscfg_boot_mode_get(void);
<b>Function descriptions</b>	get code start mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>syscfg_code_start_enum</b>	code start mode
<i>FLASH_START</i>	flash start mode
<i>SRAM_START</i>	sram start mode
<i>SYSTEM_START</i>	system start mode

Example:

```
/* get code start mode */
if(FLASH_START==syscfg_boot_mode_get()){
}
```

## 3.22. 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), 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 [3.22.1](#), the TIMER firmware functions are introduced in chapter [3.22.2](#).

### 3.22.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

**Table 3-610. TIMERx Registers**

Registers	Descriptions
TIMER_CTL0	Control register 0
TIMER_CTL1	Control register 1
TIMER_SMCFG	Slave mode configuration register
TIMER_DMAINTEN	DMA and interrupt enable register
TIMER_INTF	Interrupt flag register
TIMER_SWEVG	Software event generation register
TIMER_CHCTL0	Channel control register 0
TIMER_CHCTL1	Channel control register 1
TIMER_CHCTL2	Channel control register 2
TIMER_CNT	Counter register
TIMER_PSC	Prescaler register
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

### 3.22.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-611. TIMERx firmware function**

Function name	Function description
timer_deinit	deinit a timer
timer_struct_para_init	initialize the parameters of TIMER init parameter struct with the default values
timer_init	initialize TIMER counter
timer_enable	enable a timer
timer_disable	disable a timer
timer_auto_reload_shadow_enable	enable the auto reload shadow function
timer_auto_reload_shadow_disable	disable the auto reload shadow function
timer_update_event_enable	enable the update event
timer_update_event_disable	disable the update event
timer_counter_alignment	set TIMER counter alignment mode
timer_counter_up_direction	set TIMER counter up direction
timer_counter_down_direction	set TIMER counter down direction

Function name	Function description
timer_prescaler_config	configure TIMER prescaler
timer_repetition_value_config	configure TIMER repetition register value
timer_autoreload_value_config	configure TIMER autoreload register value
timer_counter_value_config	configure TIMER counter register value
timer_counter_read	read TIMER counter value
timer_prescaler_read	read TIMER prescaler value
timer_single_pulse_mode_config	configure TIMER single pulse mode
timer_update_source_config	configure TIMER update source
timer_dma_enable	enable the TIMER DMA
timer_dma_disable	disable the TIMER DMA
timer_channel_dma_request_source_select	channel DMA request source selection
timer_dma_transfer_config	configure the TIMER DMA transfer
timer_event_software_generate	software generate events
timer_break_struct_para_init	initialize the parameters of TIMER break parameter struct with the default values
timer_break_config	configure TIMER break function
timer_break_enable	enable TIMER break function
timer_break_disable	disable TIMER break function
timer_automatic_output_enable	enable TIMER output automatic function
timer_automatic_output_disable	disable TIMER output automatic function
timer_primary_output_config	configure TIMER primary output function
timer_channel_control_shadow_config	channel capture/compare control shadow register enable
timer_channel_control_shadow_update_config	configure TIMER channel control shadow register update control
timer_channel_output_struct_para_init	initialize the parameters of TIMER channel output parameter struct with the default values
timer_channel_output_config	configure TIMER channel output function
timer_channel_output_mode_config	configure TIMER channel output compare mode
timer_channel_output_pulse_value_config	configure TIMER channel output pulse value
timer_channel_output_shadow_config	configure TIMER channel output shadow function
timer_channel_output_fast_config	configure TIMER channel output fast function
timer_channel_output_clear_config	configure TIMER channel output clear function
timer_channel_output_polarity_config	configure TIMER channel output polarity
timer_channel_complementary_output_polarity_config	configure TIMER channel complementary output polarity
timer_channel_output_state_config	configure TIMER channel enable state

Function name	Function description
timer_channel_complementary_output_state_config	configure TIMER channel complementary output enable state
timer_channel_input_struct_para_init	initialize the parameters of TIMER channel input parameter struct with the default values
timer_input_capture_config	configure TIMER input capture parameter
timer_channel_input_capture_prescaler_config	configure TIMER channel input capture prescaler value
timer_channel_capture_value_register_read	read TIMER channel capture compare register value
timer_input_pwm_capture_config	configure TIMER input pwm capture function
timer_hall_mode_config	configure TIMER hall sensor mode
timer_input_trigger_source_select	select TIMER input trigger source
timer_master_output_trigger_source_select	select TIMER master mode output trigger source
timer_slave_mode_select	select TIMER slave mode
timer_master_slave_mode_config	configure TIMER master slave mode
timer_external_trigger_config	configure TIMER external trigger input
timer_quadrature_decoder_mode_config	configure TIMER quadrature decoder mode
timer_internal_clock_config	configure TIMER internal clock mode
timer_internal_trigger_as_external_clock_config	configure TIMER the internal trigger as external clock input
timer_external_trigger_as_external_clock_config	configure TIMER the external trigger as external clock input
timer_external_clock_mode0_config	configure TIMER the external clock mode 0
timer_external_clock_mode1_config	configure TIMER the external clock mode 1
timer_external_clock_mode1_disable	disable TIMER the external clock mode 1
timer_write_chxval_register_config	configure TIMER write CHxVAL register selection
timer_output_value_selection_config	configure TIMER output value selection
timer_flag_get	get TIMER flags
timer_flag_clear	clear TIMER flags
timer_interrupt_enable	enable the TIMER interrupt
timer_interrupt_disable	disable the TIMER interrupt
timer_interrupt_flag_get	get timer interrupt flag
timer_interrupt_flag_clear	clear TIMER interrupt flag

## Structure timer\_parameter\_struct

**Table 3-612. 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 3-613. Structure timer\_break\_parameter\_struct**

Member name	Function description
runoffstate	run mode off-state (TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE)
idloffstate	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 3-614. Structure timer\_oc\_parameter\_struct**

Member name	Function description
outputstate	channel output state (TIMER_CCX_ENABLE, TIMER_CCX_DISABLE)
outputnstate	channel complementary output state (TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE)
ocpolarity	channel output polarity (TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW)
ocnpolarity	channel complementary output polarity (TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW)
ocidlestate	idle state of channel output (TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH)

Member name	Function description
ocnidlestate	idle state of channel complementary output (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH)

### Structure timer\_ic\_parameter\_struct

**Table 3-615. 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 3-616. 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=0,1,2,5,15,16)</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 3-617. Function timer\_struct\_para\_init

Function name	timer_struct_para_init
Function prototype	void timer_struct_para_init(timer_parameter_struct* initpara);
Function descriptions	initialize the parameters of TIMER init parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
initpara	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-612. Structure timer parameter struct.</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER init parameter struct with a default value */
```

```
timer_parameter_struct timer_initpara;
```

```
timer_struct_para_init(&timer_initpara);
```

## timer\_init

The description of timer\_init is shown as below:

Table 3-618. Function timer\_init

Function name	timer_init
Function prototype	void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);
Function descriptions	initialize TIMER counter
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0,1,2,5,15,16)	TIMER peripheral selection
Input parameter{in}	
initpara	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-612. Structure timer parameter struct.</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER0 */
```

```
timer_parameter_struct timer_initpara;

timer_initpara.prescaler      = 107;

timer_initpara.alignedmode    = TIMER_COUNTER_EDGE;

timer_initpara.counterdirection = TIMER_COUNTER_UP;

timer_initpara.period         = 999;

timer_initpara.clockdivision  = TIMER_CKDIV_DIV1;

timer_initpara.repetitioncounter = 1;

timer_init(TIMER0,&timer_initpara);
```

### timer\_enable

The description of timer\_enable is shown as below:

**Table 3-619. 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</i> ( <i>x</i> =0,1,2,5,15,16)	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 3-620. 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</i> ( <i>x</i> =0,1,2,5,15,16)	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 3-621. 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</i> ( <i>x</i> =0,1,2,5,15,16)	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 3-622. 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</i> ( <i>x</i> =0,1,2,5,15,16)	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 3-623. 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</i> ( <i>x</i> =0,1,2,5,15,16)	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 3-624. 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=0,1,2,5,15,16)</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 3-625. 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=0,1,2)</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_CENTER_UP</i>	Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMEx_CHCTL0 register). Only when the counter is counting up, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_CENTER_BOTH</i>	Center-aligned and counting up/down assert mode. The counter counts under center-aligned and channel is configured in output mode (CHxMS=00 in TIMEx_CHCTL0 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_CENTER_UP);
```

### timer\_counter\_up\_direction

The description of timer\_counter\_up\_direction is shown as below:

**Table 3-626. 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=0,1,2)</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 3-627. 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, 1, 2)</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 3-628. 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, 1, 2, 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>	
-	-

Return value	
-	-

Example:

```
/* configure TIMER0 prescaler */
```

```
timer_prescaler_config(TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

### timer\_repetition\_value\_config

The description of timer\_repetition\_value\_config is shown as below:

**Table 3-629. Function timer\_repetition\_value\_config**

Function name	timer_repetition_value_config
Function prototype	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
Function descriptions	configure TIMER repetition register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0, 15, 16)</i>	TIMER peripheral selection
Input parameter{in}	
repetition	the counter repetition value (0~255)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 repetition register value */
```

```
timer_repetition_value_config(TIMER0, 98);
```

### timer\_autoreload\_value\_config

The description of timer\_autoreload\_value\_config is shown as below:

**Table 3-630. Function timer\_autoreload\_value\_config**

Function name	timer_autoreload_value_config
Function prototype	void timer_autoreload_value_config(uint32_t timer_periph, uint32_t autoreload);
Function descriptions	configure TIMER autoreload register value
Precondition	-
The called functions	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> ( <i>x</i> =0,1,2,5,15,16)	TIMER peripheral selection
Input parameter{in}	
<b>autoreload</b>	the counter auto-reload value 0~65535( <i>TIMERx</i> ( <i>x</i> =0,5,15,16)),0~4294967295( <i>TIMERx</i> ( <i>x</i> =1,2))
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER autoreload register value */
```

```
timer_autoreload_value_config(TIMER0, 3000);
```

### timer\_counter\_value\_config

The description of timer\_counter\_value\_config is shown as below:

**Table 3-631. 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>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> ( <i>x</i> =0,1,2,5,15,16)	TIMER peripheral selection
Input parameter{in}	
<b>counter</b>	the counter value0~65535( <i>TIMERx</i> ( <i>x</i> =0,5,15,16)),0~4294967295( <i>TIMERx</i> ( <i>x</i> =1,2))
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 counter register value */
```

```
timer_counter_value_config(TIMER0);
```

## timer\_counter\_read

The description of timer\_counter\_read is shown as below:

**Table 3-632. 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</i> ( <i>x</i> =0,1,2,5,15,16)	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 3-633. 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</i> ( <i>x</i> =0,1,2,5,15,16)	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 3-634. 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, uint32_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=0,1,2,5,15,16)</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 3-635. 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> =0,1,2,5,15,16)	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 3-636. 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> =0,1,2,5,15,16)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable, <i>TIMERx</i> ( <i>x</i> =0,1,2,15,16)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable, <i>TIMERx</i> ( <i>x</i> =0)

<i>TIMER_DMA_TRGD</i>	trigger DMA enable, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<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 3-637. 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> =0,1,2,5,15,16)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA disable, <i>TIMERx</i> ( <i>x</i> =0,1,2,15,16)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA disable, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA disable, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA disable, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_DMA_CMTD</i>	commutation DMA request disable, <i>TIMERx</i> ( <i>x</i> =0)
<i>TIMER_DMA_TRGD</i>	trigger DMA disable, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<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 3-638. 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</i> ( <i>x</i> =0, 1, 2, 15, 16)	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 y is sent when channel y event occurs
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	DMA request of channel y 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 3-639. 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

Input parameter{in}	
<b>dma_baseaddr</b>	DMA transfer access start address
<i>TIMER_DMACFG_DMA TA_CTL0</i>	DMA transfer address is TIMER_CTL0, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_CTL1</i>	DMA transfer address is TIMER_CTL1, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_SMCFG</i>	DMA transfer address is TIMER_SMCFG, TIMERx(x=0,1,2)
<i>TIMER_DMACFG_DMA TA_DMAINTEN</i>	DMA transfer address is TIMER_DMAINTEN, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_INTF</i>	DMA transfer address is TIMER_INTF, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_SWEVG</i>	DMA transfer address is TIMER_SWEVG, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_CHCTL0</i>	DMA transfer address is TIMER_CHCTL0, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_CHCTL1</i>	DMA transfer address is TIMER_CHCTL1, TIMERx(x=0,1,2)
<i>TIMER_DMACFG_DMA TA_CHCTL2</i>	DMA transfer address is TIMER_CHCTL2, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_CNT</i>	DMA transfer address is TIMER_CNT, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_PSC</i>	DMA transfer address is TIMER_PSC, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_CAR</i>	MA transfer address is TIMER_CAR, TIMERx(x=0,1,2,15,16)
<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=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_CH1CV</i>	DMA transfer address is TIMER_CH1CV, TIMERx(x=0,1,2)
<i>TIMER_DMACFG_DMA TA_CH2CV</i>	DMA transfer address is TIMER_CH2CV, TIMERx(x=0,1,2)
<i>TIMER_DMACFG_DMA TA_CH3CV</i>	DMA transfer address is TIMER_CH3CV, TIMERx(x=0,1,2)
<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=0,1,2,15,16)
<i>TIMER_DMACFG_DMA TA_DMATB</i>	DMA transfer address is TIMER_DMATB, TIMERx(x=0,1,2,15,16)
Input parameter{in}	

<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 3-640. 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=0,1,2,5,15,16)
<i>TIMER_EVENT_SRC_CH0G</i>	channel 0 capture or compare event generation, TIMERx(x=0,1,2,15,16)
<i>TIMER_EVENT_SRC_CH1G</i>	channel 1 capture or compare event generation, TIMERx(x=0,1,2)
<i>TIMER_EVENT_SRC_CH2G</i>	channel 2 capture or compare event generation, TIMERx(x=0,1,2)
<i>TIMER_EVENT_SRC_CH3G</i>	channel 3 capture or compare event generation, TIMERx(x=0,1,2)
<i>TIMER_EVENT_SRC_CMTG</i>	channel commutation event generation, TIMERx(x=0,15,16)
<i>TIMER_EVENT_SRC_TRIGG</i>	trigger event generation, TIMERx(x=0,1,2)

<i>TIMER_EVENT_SRC_B</i> <i>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 3-641. 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 3-613. 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 3-642. 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 3-613. 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 3-643. 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</i> ( <i>x</i> =0, 15, 16)	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 3-644. 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</i> ( <i>x</i> =0, 15, 16)	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 3-645. 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 3-646. 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 3-647. 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 3-648. 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>	
-	-
<b>Return value</b>	
-	-

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 3-649. 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 3-650. Function timer\_channel\_output\_struct\_para\_init**

Function name	timer_channel_output_struct_para_init
Function prototype	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpa);
Function descriptions	initialize the parameters of TIMER channel output parameter struct with the default values

Precondition	-
The called functions	-
Input parameter{in}	
ocpara	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-614. Structure timer_oc_parameter_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER channel output parameter struct with a default value */
```

```
timer_oc_parameter_struct timer_ocinitpara;
```

```
timer_channel_output_struct_para_init(&timer_ocinitpara);
```

### timer\_channel\_output\_config

The description of timer\_channel\_output\_config is shown as below:

**Table 3-651. Function timer\_channel\_output\_config**

Function name	timer_channel_output_config
Function prototype	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);
Function descriptions	configure TIMER channel output function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMERx(x=0,1,2,15,16))
TIMER_CH_1	TIMER channel 1 (TIMERx(x=0,1,2))
TIMER_CH_2	TIMER channel 2 (TIMERx(x=0,1,2))
TIMER_CH_3	TIMER channel 3 (TIMERx(x=0,1,2))
Input parameter{in}	
ocpara	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-614. Structure timer_oc_parameter_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 channel 0 output function */

timer_oc_parameter_struct timer_ocinitpara;

timer_ocinitpara.outputstate  = TIMER_CCX_ENABLE;

timer_ocinitpara.outputnstate = TIMER_CCXN_ENABLE;

timer_ocinitpara.ocpolarity   = TIMER_OC_POLARITY_HIGH;

timer_ocinitpara.ocnpolarity  = TIMER_OCN_POLARITY_HIGH;

timer_ocinitpara.ocidlestate  = TIMER_OC_IDLE_STATE_HIGH;

timer_ocinitpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;

timer_channel_output_config(TIMER0, TIMER_CH_0, &timer_ocinitpara);

```

### timer\_channel\_output\_mode\_config

The description of timer\_channel\_output\_mode\_config is shown as below:

**Table 3-652. 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}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
Input parameter{in}	
<b>ocmode</b>	channel output compare mode
<i>TIMER_OC_MODE_TIMING</i>	timing mode
<i>TIMER_OC_MODE_ACTIVE</i>	set the channel output
<i>TIMER_OC_MODE_INACTIVE</i>	clear the channel output

<i>TIMER_OC_MODE_TOGGLE</i>	toggle on match
<i>TIMER_OC_MODE_LOW</i>	force low mode
<i>TIMER_OC_MODE_HIGH</i>	force high mode
<i>TIMER_OC_MODE_PWM0</i>	PWM mode 0
<i>TIMER_OC_MODE_PWM1</i>	PWM mode 1
<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 3-653. 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=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
<b>Input parameter{in}</b>	
<b>pulse</b>	channel output pulse value
<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 3-654. Function timer\_channel\_output\_shadow\_config**

Function name	timer_channel_output_shadow_config
Function prototype	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
Function descriptions	configure TIMER channel output shadow function
Precondition	-
The called functions	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*configure TIMER0 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config (TIMER0, TIMER_CH_0,  
TIMER_OC_SHADOW_ENABLE);
```

## timer\_channel\_output\_fast\_config

The description of timer\_channel\_output\_fast\_config is shown as below:

**Table 3-655. 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=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
<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 3-656. 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 periphera
<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=0,1,2))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
<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 3-657. 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 oc_polarity);
<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=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
<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 3-658. 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>ocnpolarity</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 3-659. 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=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
<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 3-660. 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>ocnstate</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 3-661. 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 3-615. 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 3-662. 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=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
<b>Input parameter{in}</b>	
<b>icpara</b>	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-615. Structure timer_ic_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```

/* configure TIMER0 input capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);

```

### timer\_channel\_input\_capture\_prescaler\_config

The description of timer\_channel\_input\_capture\_prescaler\_config is shown as below:

**Table 3-663. 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}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,1,2,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,1,2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,1,2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,1,2))
Input parameter{in}	
<b>prescaler</b>	channel input capture prescaler value
<i>TIMER_IC_PSC_DIV1</i>	no prescaler
<i>TIMER_IC_PSC_DIV2</i>	divided by 2
<i>TIMER_IC_PSC_DIV4</i>	divided by 4
<i>TIMER_IC_PSC_DIV8</i>	divided by 8
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 input capture prescaler value */

timer_channel_input_capture_prescaler_config(TIMER0, TIMER_CH_0,
TIMER_IC_PSC_DIV2);
```

### timer\_channel\_capture\_value\_register\_read

The description of timer\_channel\_capture\_value\_register\_read is shown as below:

**Table 3-664. Function timer\_channel\_capture\_value\_register\_read**

Function name	timer_channel_capture_value_register_read
Function prototype	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
Function descriptions	read TIMER channel capture compare register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMERx(x=0,1,2,15,16))
TIMER_CH_1	TIMER channel 1 (TIMERx(x=0,1,2))
TIMER_CH_2	TIMER channel 2 (TIMERx(x=0,1,2))
TIMER_CH_3	TIMER channel 3 (TIMERx(x=0,1,2))
Output parameter{out}	
-	-
Return value	
uint32_t	channel capture compare register value (0~0xFFFFFFFF)

Example:

```
/* read TIMER0 channel 0 capture compare register value */

uint32_t ch0_value = 0;

ch0_value = timer_channel_capture_value_register_read(TIMER0, TIMER_CH_0);
```

### timer\_input\_pwm\_capture\_config

The description of timer\_input\_pwm\_capture\_config is shown as below:

Table 3-665. Function timer\_input\_pwm\_capture\_config

Function name	timer_input_pwm_capture_config
Function prototype	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
Function descriptions	configure TIMER input pwm capture function
Precondition	-
The called functions	timer_channel_input_capture_prescaler_config
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0,1,2)	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0
TIMER_CH_1	TIMER channel 1
Input parameter{in}	
icpwm	TIMER channel input pwm parameter struct, the structure members can refer to <a href="#">Table 3-615. Structure timer_ic_parameter_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 input pwm capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_pwm_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);

```

### timer\_hall\_mode\_config

The description of timer\_hall\_mode\_config is shown as below:

Table 3-666. Function timer\_hall\_mode\_config

Function name	timer_hall_mode_config
Function prototype	void timer_hall_mode_config(uint32_t timer_periph, uint32_t hallmode);
Function descriptions	configure TIMER hall sensor mode
Precondition	-
The called functions	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 hall sensor mode */
timer_hall_mode_config (TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

### timer\_input\_trigger\_source\_select

The description of timer\_input\_trigger\_source\_select is shown as below:

**Table 3-667. 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>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,1,2)</i>	TIMER peripheral selection
Input parameter{in}	
<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</i>	CI0 edge flag

<i>EL_CIOF_ED</i>	
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output
<i>TIMER_SMCFG_TRGS</i> <i>EL_C1FE1</i>	channel 1 input Filtered output
<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 3-668. 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=0,1,2)</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>	Update. In this mode the master mode controller selects the update event

<code>_UPDATE</code>	as TRGO.
<code>TIMER_TRI_OUT_SRC_CC0</code>	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.
<code>TIMER_TRI_OUT_SRC_O0CPRE</code>	Compare. In this mode the master mode controller selects the O0CPRE signal is used as TRGO.
<code>TIMER_TRI_OUT_SRC_O1CPRE</code>	Compare. In this mode the master mode controller selects the O1CPRE signal is used as TRGO.
<code>TIMER_TRI_OUT_SRC_O2CPRE</code>	Compare. In this mode the master mode controller selects the O2CPRE signal is used as TRGO.
<code>TIMER_TRI_OUT_SRC_O3CPRE</code>	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 3-669. 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
<code>TIMERx(x=0,1,2)</code>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>slavemode</b>	slave mode
<code>TIMER_SLAVE_MODE_DISABLE</code>	slave mode disable
<code>TIMER_QUAD_DECODER_MODE0</code>	quadrature decoder mode 0
<code>TIMER_QUAD_DECODER_MODE1</code>	quadrature decoder mode 1
<code>TIMER_QUAD_DECODER_MODE2</code>	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 3-670. 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, uint32_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=0,1,2)</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 3-671. 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=0,1,2)</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 3-672. 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=0,1,2)</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 3-673. Function timer\_internal\_clock\_config**

Function name	timer_internal_clock_config
Function prototype	void timer_internal_clock_config(uint32_t timer_periph);
Function descriptions	configure TIMER internal clock mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0,1,2)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 internal clock mode */
```

```
timer_internal_clock_config (TIMER0);
```

### timer\_internal\_trigger\_as\_external\_clock\_config

The description of timer\_internal\_trigger\_as\_external\_clock\_config is shown as below:

**Table 3-674. Function timer\_internal\_trigger\_as\_external\_clock\_config**

Function name	timer_internal_trigger_as_external_clock_config
Function prototype	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
Function descriptions	configure TIMER the internal trigger as external clock input
Precondition	-
The called functions	timer_input_trigger_source_select
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0,1,2)	TIMER peripheral selection
Input parameter{in}	
intrigger	trigger selection
TIMER_SMCFG_TRGS	Internal trigger input 0 (ITI0)

<i>EL_ITI0</i>	
<i>TIMER_SMCFG_TRGS</i> <i>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 3-675. 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=0,1,2)</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>	active low or falling edge active

<i>FALLING</i>	
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	active both edge
<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 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 3-676. Function timer\_external\_clock\_mode0\_config**

<b>Function name</b>	timer_external_clock_mode0_config
<b>Function prototype</b>	void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external clock mode0
<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=0,1,2)</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

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 3-677. Function timer\_external\_clock\_mode1\_config**

Function name	timer_external_clock_mode1_config
Function prototype	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t extpolarity, uint32_t extfilter);
Function descriptions	configure TIMER the external clock mode1
Precondition	-
The called functions	timer_external_trigger_config
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,1,2)</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>extpolarity</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 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 3-678. 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=0,1,2)</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 3-679. 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</i> ( <i>x</i> =0, 1, 2, 15, 16)	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 3-680. 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</i> ( <i>x</i> =0, 15, 16)	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 3-681. 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=0,1,2,5,15,16)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, TIMERx(x=0,1,2,15,16)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, TIMERx(x=0,15,16)
<i>TIMER_FLAG_TRG</i>	trigger flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_BRK</i>	break flag, TIMERx(x=0,15,16)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, TIMERx(x=0,1,2,15,16)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, TIMERx(x=0,1,2)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, TIMERx(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TIMER0 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get(TIMER0, TIMER_FLAG_UP);
```

## timer\_flag\_clear

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

Table 3-682. Function timer\_flag\_clear

Function name	timer_flag_clear
Function prototype	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
Function descriptions	clear TIMER flags
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
flag	the timer interrupt flags
TIMER_FLAG_UP	update flag, TIMERx(x=0,1,2,5,15,16)
TIMER_FLAG_CH0	channel 0 flag, TIMERx(x=0,1,2,15,16)
TIMER_FLAG_CH1	channel 1 flag, TIMERx(x=0,1,2)
TIMER_FLAG_CH2	channel 2 flag, TIMERx(x=0,1,2)
TIMER_FLAG_CH3	channel 3 flag, TIMERx(x=0,1,2)
TIMER_FLAG_CMT	channel commutation flag, TIMERx(x=0,15,16)
TIMER_FLAG_TRG	trigger flag, TIMERx(x=0,1,2)
TIMER_FLAG_BRK	break flag, TIMERx(x=0,15,16)
TIMER_FLAG_CH0O	channel 0 overcapture flag, TIMERx(x=0,1,2,15,16)
TIMER_FLAG_CH1O	channel 1 overcapture flag, TIMERx(x=0,1,2)
TIMER_FLAG_CH2O	channel 2 overcapture flag, TIMERx(x=0,1,2)
TIMER_FLAG_CH3O	channel 3 overcapture flag, TIMERx(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TIMER0 update flags */
```

```
timer_flag_clear(TIMER0, TIMER_FLAG_UP);
```

### timer\_interrupt\_enable

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

Table 3-683. Function timer\_interrupt\_enable

Function name	timer_interrupt_enable
Function prototype	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
Function descriptions	enable the TIMER interrupt
Precondition	-
The called functions	-
Input parameter{in}	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	timer interrupt enable source
<i>TIMER_INT_UP</i>	update interrupt enable, TIMERx (x=0,1,2,5,15,16)
<i>TIMER_INT_CH0</i>	channel 0 interrupt enable, TIMERx (x=0,1,2,15,16)
<i>TIMER_INT_CH1</i>	channel 1 interrupt enable, TIMERx (x=0,1,2)
<i>TIMER_INT_CH2</i>	channel 2 interrupt enable, TIMERx (x=0,1,2)
<i>TIMER_INT_CH3</i>	channel 3 interrupt enable , TIMERx (x=0,1,2)
<i>TIMER_INT_CMT</i>	commutation interrupt enable, TIMERx (x=0,15,16)
<i>TIMER_INT_TRG</i>	trigger interrupt enable, TIMERx (x=0,1,2)
<i>TIMER_INT_BRK</i>	break interrupt enable, TIMERx (x=0,15,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-684. 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>	-
<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>interrupt</b>	timer interrupt disable source
<i>TIMER_INT_UP</i>	update interrupt disable, TIMERx(x=0,1,2,5,15,16)
<i>TIMER_INT_CH0</i>	channel 0 interrupt disable, TIMERx(x=0,1,2,15,16)
<i>TIMER_INT_CH1</i>	channel 1 interrupt disable, TIMERx(x=0,1,2)
<i>TIMER_INT_CH2</i>	channel 2 interrupt disable, TIMERx(x=0,1,2)
<i>TIMER_INT_CH3</i>	channel 3 interrupt disable, TIMERx(x=0,1,2)
<i>TIMER_INT_CMT</i>	commutation interrupt disable, TIMERx(x=0,15,16)
<i>TIMER_INT_TRG</i>	trigger interrupt disable, TIMERx(x=0,1,2)

<i>TIMER_INT_BRK</i>	break interrupt disable, <i>TIMERx</i> ( <i>x</i> =0,15,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-685. 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> =0,1,2,5,15,16)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,1,2,15,16)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<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> =0,1,2)
<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 3-686. 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> =0,1,2,5,15,16)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,1,2,15,16)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<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> =0,1,2)
<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);
```

## 3.23. TRNG

The true random number generator (TRNG) module can generate a 32-bit value using continuous analog noise. The TRNG registers are listed in chapter [3.23.1](#). the TRNG firmware functions are introduced in chapter [3.23.2](#).

### 3.23.1. Descriptions of Peripheral registers

TRNG registers are listed in the table shown as below:

**Table 3-687 TRNG Registers**

Registers	Descriptions
TRNG_CTL	TRNG control register
TRNG_STAT	TRNG status register
TRNG_DATA	TRNG data register

### 3.23.2. Descriptions of Peripheral functions

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

**Table 3-688. TRNG firmware function**

Function name	Function description
trng_deinit	deinitialize the TRNG
trng_enable	enable the TRNG interface
trng_disable	disable the TRNG interface
trng_get_true_random_data	get the true random data
trng_interrupt_enable	enable the TRNG interrupt
trng_interrupt_disable	disable the TRNG interrupt
trng_flag_get	get the TRNG status flags
trng_interrupt_flag_get	get the TRNG interrupt flags
trng_interrupt_flag_clear	clear the TRNG interrupt flags

#### Enum trng\_flag\_enum

**Table 3-689. 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 3-690. 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 3-691. Function trng\_deinit

Function name	trng_deinit
Function prototype	void trng_deinit(void)
Function descriptions	deinit the TRNG
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset TRNG */
trng_deinit();
```

### trng\_enable

The description of trng\_enable is shown as below:

Table 3-692. Function trng\_enable

Function name	trng_enable
Function prototype	void trng_enable(void)
Function descriptions	enable the TRNG interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TRNG */
trng_enable();
```

### trng\_disable

The description of trng\_disable is shown as below:

Table 3-693 Function trng\_disable

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

Example:

```
/* disable TRNG */
trng_disable();
```

### trng\_get\_true\_random\_data

The description of trng\_get\_true\_random\_data is shown as below:

Table 3-694 Function trng\_get\_true\_random\_data

Function name	trng_get_true_random_data
Function prototype	uint32_t trng_get_true_random_data(void)
Function descriptions	get the true random data
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the generated random data

Example:

```
/* get true random data */
uint32_t data;
data = trng_get_true_random_data();
```

### trng\_interrupt\_enable

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

Table 3-695 Function trng\_interrupt\_enable

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

Example:

```
/* enable TRNG interrupt */
trng_interrupt_enable();
```

### trng\_interrupt\_disable

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

Table 3-696 Function trng\_interrupt\_disable

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

Example:

```
/* disable TRNG interrupt */
trng_interrupt_disable();
```

### trng\_flag\_get

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

Table 3-697 Function `trng_flag_get`

Function name	<code>trng_flag_get</code>
Function prototype	<code>FlagStatus trng_flag_get(trng_flag_enum flag);</code>
Function descriptions	get the trng status flags
Precondition	-
The called functions	-
Input parameter{in}	
flag	<code>trng_flag_enum</code> , please refer to <a href="#">Table 3-689. Enum <code>trng_flag_enum</code></a>
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get TRNG clock error current flag status */

FlagStatus flag_status = RESET;

flag_status == trng_flag_get(TRNG_FLAG_CECS);
```

### `trng_interrupt_flag_get`

The description of `trng_interrupt_flag_get` is shown as below:

Table 3-698 Function `trng_interrupt_flag_get`

Function name	<code>trng_interrupt_flag_get</code>
Function prototype	<code>FlagStatus trng_interrupt_flag_get(trng_int_flag_enum int_flag);</code>
Function descriptions	get the trng interrupt flags
Precondition	-
The called functions	-
Input parameter{in}	
flag	<code>trng_int_flag_enum</code> , please refer to <a href="#">Table 3-690. Enum <code>trng_int_flag_enum</code></a>
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get TRNG clock error interrupt flag */

FlagStatus interrupt_flag = RESET;

interrupt_flag = trng_interrupt_flag_get(TRNG_INT_FLAG_CEIF);
```

## trng\_interrupt\_flag\_clear

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

**Table 3-699 Function trng\_interrupt\_flag\_clear**

<b>Function name</b>	trng_interrupt_flag_clear
<b>Function prototype</b>	void trng_interrupt_flag_clear(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 3-690. 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);
```

## 3.24. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.24.1](#), the USART firmware functions are introduced in chapter [3.24.2](#).

### 3.24.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

**Table 3-700. 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

Registers	Descriptions
USART_TDATA	Transmit data register
USART_CHC	Coherence control register
USART_RFCS	Receive FIFO control and status register

### 3.24.2. Descriptions of Peripheral functions

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

**Table 3-701. 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
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_command_enable	enable USART command
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

Function name	Function description
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
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_flag_get	get flag in STAT/RFCs register

Function name	Function description
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 3-702. 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
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 3-703. 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

Member name	Function description
USART_INT_FLAG_PERR	parity error interrupt and flag
USART_INT_FLAG_TBE	transmitter buffer empty interrupt and flag
USART_INT_FLAG_TC	transmission complete interrupt and flag
USART_INT_FLAG_RBNE	read data buffer not empty interrupt and flag
USART_INT_FLAG_RBNE_ORE RR	read data buffer not empty interrupt and overrun error flag
USART_INT_FLAG_IDLE	IDLE line detected interrupt and flag
USART_INT_FLAG_LBD	LIN break detected interrupt and flag
USART_INT_FLAG_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_ORER R	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 3-704. Enum usart\_interrupt\_enum**

Member name	Function description
USART_INT_EB	end of block interrupt
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 3-705. 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

Member name	Function description
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 3-706. 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>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=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 3-707. 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>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>baudval</b>	baud rate value
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* configure USART0 baud rate value */
usart_baudrate_set(USART0, 115200);
```

### usart\_parity\_config

The description of usart\_parity\_config is shown as below:

**Table 3-708. 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 function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
USART0/UARTx	x=1,2
<b>Input parameter{in}</b>	
<b>paritycfg</b>	USART parity configure
USART_PM_NONE	no parity
USART_PM_ODD	odd parity
USART_PM_EVEN	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 3-709. 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>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>wlen</b>	USART word length configure
<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 3-710. 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>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>stblen</b>	USART stop bit configure
<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 3-711. 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>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=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 3-712. 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>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=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 3-713. 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>	USART0/UARTx peripheral
USART0/UARTx	x=1,2
<b>Input parameter{in}</b>	
<b>txconfig</b>	enable or disable USART transmitter
USART_TRANSMIT_ENABLE	enable USART transmission
USART_TRANSMIT_DISABLE	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 3-714. 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>	USART0/UARTx peripheral

USART0/UARTx	x=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 3-715. 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>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
USART0/UARTx	x=1,2
<b>Input parameter{in}</b>	
<b>msbf</b>	LSB/MSB
USART_MSBF_LSB	LSB first
USART_MSBF_MSB	MSB first
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-716. 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>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
USART0/UARTx	x=1,2
<b>Input parameter{in}</b>	
<b>invertpara</b>	refer to <a href="#">Table 3-705. Enum usart_invert_enum</a>
USART_DINV_ENAB LE	data bit level inversion
USART_DINV_DISAB LE	data bit level not inversion
USART_TXPIN_ENAB LE	TX pin level inversion
USART_TXPIN_DISAB LE	TX pin level not inversion
USART_RXPIN_ENAB LE	RX pin level inversion
USART_RXPIN_DISAB LE	RX pin level not inversion
USART_SWAP_ENAB LE	swap TX/RX pins
USART_SWAP_DISAB LE	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\_oversize\_enable

The description of usart\_oversize\_enable is shown as below:

**Table 3-717. Function usart\_oversize\_enable**

<b>Function name</b>	usart_oversize_enable
<b>Function prototype</b>	void usart_oversize_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable the USART oversize function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the USART0 oversize function */
```

```
usart_oversize_enable(USART0);
```

## usart\_oversize\_disable

The description of usart\_oversize\_disable is shown as below:

**Table 3-718. Function usart\_oversize\_disable**

<b>Function name</b>	usart_oversize_disable
<b>Function prototype</b>	void usart_oversize_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable the USART oversize function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the USART0 oversize function */
```

```
usart_oversize_disable(USART0);
```

## usart\_oversample\_config

The description of usart\_oversample\_config is shown as below:

**Table 3-719. 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>	USART0/UARTx peripheral
USART0/UARTx	x=1,2
<b>Input parameter{in}</b>	
<b>oversamp</b>	oversample value
USART_OVSMOD_8	8 bits
USART_OVSMOD_16	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 3-720. Function usart\_sample\_bit\_config**

<b>Function name</b>	usart_sample_bit_config
<b>Function prototype</b>	void usart_sample_bit_config(uint32_t usart_periph, uint32_t osb);
<b>Function descriptions</b>	configure the sample bit method
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
USART0/UARTx	x=1,2
<b>Input parameter{in}</b>	
<b>osb</b>	sample bit
USART_OSB_1bit	1 bits
USART_OSB_3bit	3 bits
<b>Output parameter{out}</b>	

-	-
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 3-721. Function usart\_receiver\_timeout\_enable**

<b>Function name</b>	usart_receiver_timeout_enable
<b>Function prototype</b>	void usart_receiver_timeout_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable receiver timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-722. 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>	USART0/UARTx peripheral
<i>USART0</i>	USART0

Output parameter{out}	
-	-
Return value	
-	-

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 3-723. Function usart\_receiver\_timeout\_threshold\_config**

Function name	usart_receiver_timeout_threshold_config
Function prototype	void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);
Function descriptions	configure receiver timeout threshold
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0	USART0
Input parameter{in}	
rtimeout	receiver timeout threshold (0x00000000 - 0x00FFFFFF)
Output parameter{out}	
-	-
Return value	
-	-

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 3-724. Function usart\_data\_transmit**

Function name	usart_data_transmit
Function prototype	void usart_data_transmit(uint32_t usart_periph, uint32_t data);
Function descriptions	USART transmit data function
Precondition	-

The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Input parameter{in}	
data	data of transmission
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 transmit data */
```

```
usart_data_transmit(USART0, 0xAA);
```

### usart\_data\_receive

The description of usart\_data\_receive is shown as below:

**Table 3-725. Function usart\_data\_receive**

Function name	usart_data_receive
Function prototype	uint16_t usart_data_receive(uint32_t usart_periph);
Function descriptions	USART receive data function
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Output parameter{out}	
-	-
Return value	
uint16_t	data of received

Example:

```
/* USART0 receive data */
```

```
uint16_t temp;
```

```
temp = usart_data_receive(USART0);
```

### usart\_command\_enable

The description of usart\_command\_enable is shown as below:

Table 3-726. Function `usart_command_enable`

<b>Function name</b>	<code>usart_command_enable</code>
<b>Function prototype</b>	<code>void usart_command_enable(uint32_t usart_periph, uint32_t cmdtype);</code>
<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>USART0/UARTx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>cmdtype</b>	command type
<i>USART_CMD_SBKCMD</i> <i>D</i>	send break command
<i>USART_CMD_MMCMMD</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\_address\_config**

The description of `usart_address_config` is shown as below:

Table 3-727. Function `usart_address_config`

<b>Function name</b>	<code>usart_address_config</code>
<b>Function prototype</b>	<code>void usart_address_config(uint32_t usart_periph, uint8_t addr);</code>
<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>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>addr</b>	address of USART terminal(0x00-0xFF)
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

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 3-728. 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>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>addmod</b>	address detection mode
<i>USART_ADDDM_4BIT</i>	4 bits
<i>USART_ADDDM_FULLBIT</i>	full bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure address detection mode */
```

```
usart_address_config(USART0, USART_ADDDM_4BIT);
```

### usart\_mute\_mode\_enable

The description of usart\_mute\_mode\_enable is shown as below:

**Table 3-729. 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>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-730. 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>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=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 3-731. 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>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=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 3-732. 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>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<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 3-733. 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>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<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 3-734. 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 detection length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<b>Input parameter{in}</b>	
<b>lblen</b>	LIN break detection length
<i>USART_LBLEN_10B</i>	10 bits break detection
<i>USART_LBLEN_11B</i>	11 bits break detection
<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 3-735. 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>	USART0/UARTx peripheral
USART0/UARTx	x=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 3-736. 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>	USART0/UARTx peripheral
USART0/UARTx	x=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 3-737. 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 clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<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 3-738. 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 clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<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 3-739. 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>	USART0/UARTx peripheral
USART0	USART0
<b>Input parameter{in}</b>	
<b>clen</b>	CK length
USART_CLEN_NONE	clock pulse of the last data bit (MSB) is not output to the CK pin
USART_CLEN_EN	clock pulse of the last data bit (MSB) is output to the CK pin
<b>Input parameter{in}</b>	
<b>cph</b>	clock phase
USART_CPH_1CK	first clock transition is the first data capture edge
USART_CPH_2CK	second clock transition is the first data capture edge
<b>Input parameter{in}</b>	
<b>cpl</b>	clock polarity
USART_CPL_LOW	steady low value on CK pin
USART_CPL_HIGH	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 3-740. Function `usart_guard_time_config`

<b>Function name</b>	<code>usart_guard_time_config</code>
<b>Function prototype</b>	<code>void usart_guard_time_config(uint32_t usart_periph, uint32_t gaut);</code>
<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>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<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 3-741. Function `usart_smartcard_mode_enable`

<b>Function name</b>	<code>usart_smartcard_mode_enable</code>
<b>Function prototype</b>	<code>void usart_smartcard_mode_enable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	enable smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<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 3-742. 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>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<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 3-743. 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>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<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 3-744. 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>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<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 3-745. 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>USART0</i>	USART0
<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 3-746. 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>USART0</i>	USART0
<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 3-747. 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>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<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 3-748. 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>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<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 3-749. 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>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<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 3-750. 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>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<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 3-751. 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 or smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<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 3-752. 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>	USART0/UARTx peripheral
<i>USART0</i>	USART0
<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 3-753. 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>	USART0/UARTx peripheral
USART0/UARTx	x=1,2
<b>Input parameter{in}</b>	
<b>rtsconfig</b>	enable or disable RTS
USART_RTS_ENABLE	enable RTS
USART_RTS_DISABLE	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 3-754. Function usart\_hardware\_flow\_cts\_config**

<b>Function name</b>	usart_hardware_flow_cts_config
<b>Function prototype</b>	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
<b>Function descriptions</b>	configure hardware flow control CTS
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
USART0/UARTx	x=1,2
<b>Input parameter{in}</b>	
<b>ctsconfig</b>	enable or disable CTS
USART_CTS_ENABLE	enable CTS
USART_CTS_DISABLE	disable CTS
<b>Output parameter{out}</b>	
-	-

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 3-755. 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	USART0/UARTx peripheral
USART0/UARTx	x=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 3-756. Function usart\_rs45\_driver\_enable**

Function name	usart_rs485_driver_enable
Function prototype	void usart_rs485_driver_enable(uint32_t usart_periph);
Function descriptions	enable RS485 driver
Precondition	-

The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

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 3-757. Function usart\_rs45\_driver\_disable**

Function name	usart_rs485_driver_disable
Function prototype	void usart_rs485_driver_disable(uint32_t usart_periph);
Function descriptions	disable RS485 driver
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USART0/UARTx peripheral
USART0/UARTx	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 RS485 driver */
usart_rs485_driver_disable(USART0);
```

### usart\_driver\_assertime\_config

The description of usart\_driver\_assertime\_config is shown as below:

**Table 3-758. Function usart\_driver\_assertime\_config**

Function name	usart_driver_assertime_config
Function prototype	void usart_driver_assertime_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>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>deatime</b>	driver enable assertion time(0x00-0x1F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set USART0 driver assertime */
usart_driver_assertime_config(USART0, 0x1F);
```

### usart\_driver\_deassertime\_config

The description of usart\_driver\_deassertime\_config is shown as below:

**Table 3-759. Function usart\_driver\_deassertime\_config**

<b>Function name</b>	usart_driver_deassertime_config
<b>Function prototype</b>	void usart_driver_deassertime_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>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>deatime</b>	driver enable deassertime (0x00-0x1F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set USART0 driver deassertime */
usart_driver_deassertime_config(USART0, 0x1F);
```

## usart\_depolarity\_config

The description of usart\_depolarity\_config is shown as below:

**Table 3-760. 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>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
USART0/UARTx	x=1,2
<b>Input parameter{in}</b>	
<b>dep</b>	DE signal
USART_DEP_HIGH	DE signal is active high
USART_DEP_LOW	DE signal is active low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-761. 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>	USART0/UARTx peripheral
USART0/UARTx	x=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

<i>A_DISABLE</i>	
<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 3-762. 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>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>dmacmd</b>	USART DMA mode
<i>USART_TRANSMIT_DMA_ENABLE</i>	enable USART DMA for transmission
<i>USART_TRANSMIT_DMA_DISABLE</i>	disable USART DMA for transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-763. Function `usart_reception_error_dma_disable`

<b>Function name</b>	<code>usart_reception_error_dma_disable</code>
<b>Function prototype</b>	<code>void usart_reception_error_dma_disable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	disable DMA on reception error
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-764. Function `usart_reception_error_dma_enable`

<b>Function name</b>	<code>usart_reception_error_dma_enable</code>
<b>Function prototype</b>	<code>void usart_reception_error_dma_enable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	enable DMA on reception error
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-765. 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
USART0	USART0
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 3-766. 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
USART0	USART0
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 3-767. Function `usart_wakeup_mode_config`

<b>Function name</b>	<code>usart_wakeup_mode_config</code>
<b>Function prototype</b>	<code>void usart_wakeup_mode_config(uint32_t usart_periph, uint32_t wum);</code>
<b>Function descriptions</b>	configure the USART wakeup mode from deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	usart peripheral
<i>USART0</i>	USART0
<b>Input parameter{in}</b>	
<b>wum</b>	wakeup mode
<i>USART_WUM_ADDR</i>	WUF active on address match
<i>USART_WUM_START</i> <i>B</i>	WUF active on start bit
<i>USART_WUM_RBNE</i>	WUF active on RBNE
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-768. Function `usart_receive_fifo_enable`

<b>Function name</b>	<code>usart_receive_fifo_enable</code>
<b>Function prototype</b>	<code>void usart_receive_fifo_enable(uint32_t usart_periph);</code>
<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>USART0/UARTx</i>	x=1,2
<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 3-769. 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
USART0/UARTx	x=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 3-770. 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
USART0/UARTx	x=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\_flag\_get

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

**Table 3-771. 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>	USART0/UARTx peripheral
USART0/UARTx	x=1,2
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to <a href="#">Table 3-702. 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

<i>USART_FLAG_RFF</i>	receive FIFO full flag
<i>USART_FLAG_RFFINT</i>	receive FIFO full interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-772. 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>	
<b>usart_periph</b>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to <a href="#">Table 3-702. Enum usart_flag_enum</a>
<i>USART_FLAG_PERR</i>	parity error flag
<i>USART_FLAG_FERR</i>	frame error flag
<i>USART_FLAG_NERR</i>	noise detected flag
<i>USART_FLAG_ORER</i> <i>R</i>	overrun error flag
<i>USART_FLAG_IDLE</i>	idle line detected flag
<i>USART_FLAG_TC</i>	transmission complete flag
<i>USART_FLAG_LBD</i>	LIN break detected flag
<i>USART_FLAG_CTSF</i>	CTS change flag
<i>USART_FLAG_RT</i>	receiver timeout flag
<i>USART_FLAG_EB</i>	end of block flag
<i>USART_FLAG_AM</i>	address match flag
<i>USART_FLAG_WU</i>	wakeup from deep-sleep mode flag
<i>USART_FLAG_EPERR</i>	early parity error flag
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* clear USART0 flag */
```

```
usart_flag_clear(USART0, USART_FLAG_TC);
```

### usart\_interrupt\_enable

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

**Table 3-773. 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>	USART0/UARTx peripheral
USART0/UARTx	x=1,2
<b>Input parameter{in}</b>	
<b>interrupt</b>	USART interrupt, refer to <a href="#">Table 3-704. Enum usart_interrupt_enum</a>
USART_INT_IDLE	idle interrupt
USART_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt enable interrupt
USART_INT_TC	transmission complete interrupt
USART_INT_TBE	transmit data register empty interrupt
USART_INT_PERR	parity error interrupt
USART_INT_AM	address match interrupt
USART_INT_RT	receiver timeout interrupt
USART_INT_EB	end of block interrupt
USART_INT_LBD	LIN break detection interrupt
USART_INT_ERR	error interrupt enable in multibuffer communication
USART_INT_CTS	CTS interrupt
USART_INT_WU	wakeup from deep-sleep mode interrupt
USART_INT_RFF	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 3-774. 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>	USART0/UARTx peripheral
USART0/UARTx	x=1,2
<b>Input parameter{in}</b>	
<b>intterrupt</b>	USART interrupt flag, refer to <a href="#">Table 3-704. Enum usart_interrupt_enum</a>
USART_INT_IDLE	idle interrupt
USART_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt enable interrupt
USART_INT_TC	transmission complete interrupt
USART_INT_TBE	transmit data register empty interrupt
USART_INT_PERR	parity error interrupt
USART_INT_AM	address match interrupt
USART_INT_RT	receiver timeout interrupt
USART_INT_EB	end of block interrupt
USART_INT_LBD	LIN break detection interrupt
USART_INT_ERR	error interrupt enable in multibuffer communication
USART_INT_CTS	CTS interrupt
USART_INT_WU	wakeup from deep-sleep mode interrupt
USART_INT_RFF	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 3-775. 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>	USART0/UARTx peripheral
USART0/UARTx	x=1,2
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag, refer to <a href="#">Table 3-703. 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	error interrupt and frame error flag

<i>R_FERR</i>	
<i>USART_INT_FLAG_RFRF</i>	receive FIFO full interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-776. Function usart\_interrupt\_flag\_clear**

<b>Function name</b>	usart_interrupt_flag_clear
<b>Function prototype</b>	void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	clear USART interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USART0/UARTx peripheral
<i>USART0/UARTx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag, refer to <a href="#">Table 3-703. Enum usart_interrupt_flag_enum</a>
<i>USART_INT_FLAG_PERR</i>	parity error flag
<i>USART_INT_FLAG_ERFERR</i>	frame error flag
<i>USART_INT_FLAG_ERNERR</i>	noise detected flag
<i>USART_INT_FLAG_RBNE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>USART_INT_FLAG_ERORERR</i>	error interrupt and overrun error
<i>USART_INT_FLAG_IDLE</i>	idle line detected flag
<i>USART_INT_FLAG_TC</i>	transmission complete flag

<i>USART_INT_FLAG_LB</i> <i>D</i>	LIN break detected flag
<i>USART_INT_FLAG_CT</i> <i>S</i>	CTS change flag
<i>USART_INT_FLAG_RT</i>	receiver timeout flag
<i>USART_INT_FLAG_EB</i>	end of block flag
<i>USART_INT_FLAG_A</i> <i>M</i>	address match flag
<i>USART_INT_FLAG_W</i> <i>U</i>	wakeup from deep-sleep mode flag
<i>USART_INT_FLAG_RF</i> <i>F</i>	receive FIFO full interrupt and flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the USART0 interrupt enable bit status */
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

## 3.25. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.25.1](#), the WWDGT firmware functions are introduced in chapter [3.25.2](#).

### 3.25.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

**Table 3-777. WWDGT Registers**

Registers	Descriptions
WWDGT_CTL	Control register
WWDGT_CFG	Configuration register
WWDGT_STAT	Status register

### 3.25.2. Descriptions of Peripheral functions

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

Table 3-778. 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 3-779. 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 3-780. 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 WWDGT counter */
```

```
wwdgt_enable();
```

### wwdgt\_counter\_update

The description of wwdgt\_counter\_update is shown as below:

**Table 3-781. 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 3-782. 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
<i>WWDGT_CFG_PSC_D</i> <i>IV1</i>	the time base of window watchdog counter = (PCLK1/4096)/1
<i>WWDGT_CFG_PSC_D</i> <i>IV2</i>	the time base of window watchdog counter = (PCLK1/4096)/2
<i>WWDGT_CFG_PSC_D</i> <i>IV4</i>	the time base of window watchdog counter = (PCLK1/4096)/4
<i>WWDGT_CFG_PSC_D</i> <i>IV8</i>	the time base of window watchdog counter = (PCLK1/4096)/8
<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(127, 80, WWDGT_CFG_PSC_DIV8);
```

### wwdgt\_interrupt\_enable

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

**Table 3-783. 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 3-784. 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 3-785. 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	Jul.20, 2023
1.1	Add SES project relevant descriptions	Jan.16, 2024

## 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.