

**GigaDevice Semiconductor Inc.**

**GD32F403xx**

**ARM® Cortex™-M4 32-bit MCU**

**Firmware Library  
User Guide**

Revision 1.2

(Mar. 2024)

# Table of Contents

<b>Table of Contents .....</b>	<b>1</b>
<b>List of Figures .....</b>	<b>4</b>
<b>List of Tables .....</b>	<b>5</b>
<b>1. Introduction .....</b>	<b>21</b>
<b>1.1. Rules of User Manual and Firmware Library .....</b>	<b>21</b>
1.1.1. Peripherals.....	21
1.1.2. Naming rules.....	22
<b>2. Firmware Library Overview.....</b>	<b>23</b>
<b>2.1. File Structure of Firmware Library .....</b>	<b>23</b>
2.1.1. Examples Folder .....	24
2.1.2. Firmware Folder.....	24
2.1.3. Template Folder .....	24
2.1.4. Utilities Folder .....	27
<b>2.2. File descriptions of Firmware Library .....</b>	<b>28</b>
<b>3. Firmware Library of Standard Peripherals .....</b>	<b>29</b>
<b>3.1. Overview of Firmware Library of Standard Peripherals.....</b>	<b>29</b>
<b>3.2. ADC .....</b>	<b>29</b>
3.2.1. Descriptions of Peripheral registers.....	29
3.2.2. Descriptions of Peripheral functions .....	30
<b>3.3. BKP.....</b>	<b>65</b>
3.3.1. Descriptions of Peripheral registers.....	65
3.3.2. Descriptions of Peripheral functions .....	66
<b>3.4. CAN .....</b>	<b>80</b>
3.4.1. Descriptions of Peripheral registers.....	80
3.4.2. Descriptions of Peripheral functions .....	81
<b>3.5. CRC .....</b>	<b>115</b>
3.5.1. Descriptions of Peripheral registers.....	115
3.5.2. Descriptions of Peripheral functions .....	115
<b>3.6. CTC.....</b>	<b>120</b>
3.6.1. Descriptions of Peripheral registers.....	120
3.6.2. Descriptions of Peripheral functions .....	121
<b>3.7. DAC .....</b>	<b>137</b>
3.7.1. Descriptions of Peripheral registers.....	错误!未定义书签。
3.7.2. Descriptions of Peripheral functions .....	错误!未定义书签。

<b>3.8. DBG .....</b>	<b>152</b>
3.8.1. Descriptions of Peripheral registers .....	152
3.8.2. Descriptions of Peripheral functions .....	152
<b>3.9. DMA .....</b>	<b>159</b>
3.9.1. Descriptions of Peripheral registers .....	159
3.9.2. Descriptions of Peripheral functions .....	160
<b>3.10. EXMC .....</b>	<b>185</b>
3.10.1. Descriptions of Peripheral registers .....	186
3.10.2. Descriptions of Peripheral functions .....	186
<b>3.11. EXTI.....</b>	<b>210</b>
3.11.1. Descriptions of Peripheral registers .....	210
3.11.2. Descriptions of Peripheral functions .....	210
<b>3.12. FMC .....</b>	<b>218</b>
3.12.1. Descriptions of Peripheral registers .....	218
3.12.2. Descriptions of Peripheral functions .....	218
<b>3.13. FWDGT.....</b>	<b>247</b>
3.13.1. Descriptions of Peripheral registers .....	247
3.13.2. Descriptions of Peripheral functions .....	248
<b>3.14. GPIO.....</b>	<b>252</b>
3.14.1. Descriptions of Peripheral registers .....	252
3.14.2. Descriptions of Peripheral functions .....	253
<b>3.15. I2C .....</b>	<b>271</b>
3.15.1. Descriptions of Peripheral registers .....	271
3.15.2. Descriptions of Peripheral functions .....	271
<b>3.16. MISC.....</b>	<b>299</b>
3.16.1. Descriptions of Peripheral registers .....	299
3.16.2. Descriptions of Peripheral functions .....	300
<b>3.17. PMU.....</b>	<b>306</b>
3.17.1. Descriptions of Peripheral registers .....	307
3.17.2. Descriptions of Peripheral functions .....	307
<b>3.18. RCU .....</b>	<b>321</b>
3.18.1. Descriptions of Peripheral registers .....	322
3.18.2. Descriptions of Peripheral functions .....	322
<b>3.19. RTC .....</b>	<b>360</b>
3.19.1. Descriptions of Peripheral registers .....	360
3.19.2. Descriptions of Peripheral functions .....	361
<b>3.20. SDIO .....</b>	<b>371</b>
3.20.1. Descriptions of Peripheral registers .....	371
3.20.2. Descriptions of Peripheral functions .....	372

<b>3.21. SPI.....</b>	<b>411</b>
3.21.1. Descriptions of Peripheral registers.....	411
3.21.2. Descriptions of Peripheral functions .....	411
<b>3.22. TIMER.....</b>	<b>445</b>
3.22.1. Descriptions of Peripheral registers.....	445
3.22.2. Descriptions of Peripheral functions .....	446
<b>3.23. USART.....</b>	<b>518</b>
3.23.1. Descriptions of Peripheral registers.....	518
3.23.2. Descriptions of Peripheral functions .....	518
<b>3.24. WWDGT.....</b>	<b>561</b>
3.24.1. Descriptions of Peripheral registers.....	561
3.24.2. Descriptions of Peripheral functions .....	561
<b>4. Revision history.....</b>	<b>568</b>



## List of Figures

Figure 2-1. File structure of firmware library of GD32F403 .....	23
Figure 2-2. Select peripheral example files .....	25
Figure 2-3. Copy the peripheral example files .....	26
Figure 2-4. Open the project file .....	26
Figure 2-5. Configure project files .....	27
Figure 2-6. Compile-debug-download .....	27

## List of Tables

Table 1-1. Peripherals .....	21
Table 2-1. Function descriptions of Firmware Library .....	28
Table 3-1. Peripheral function format of Firmware Library .....	29
Table 3-2. ADC Registers .....	30
Table 3-3. ADC firmware function .....	30
Table 3-4. Function adc_deinit .....	32
Table 3-5. Function adc_enable .....	32
Table 3-6. Function adc_disable .....	33
Table 3-7. Function adc_calibration_enable .....	34
Table 3-8. Function adc_dma_mode_enable .....	34
Table 3-9. Function adc_dma_mode_disable .....	35
Table 3-10. Function adc_tempsensor_vrefint_enable .....	36
Table 3-11. Function adc_tempsensor_vrefint_disable .....	36
Table 3-12. Function adc_resolution_config .....	37
Table 3-13. Function adc_discontinuous_mode_config .....	38
Table 3-14. Function adc_mode_config .....	39
Table 3-15. Function adc_special_function_config .....	40
Table 3-16. Function adc_data_alignment_config .....	41
Table 3-17. Function adc_channel_length_config .....	42
Table 3-18. Function adc_regular_channel_config .....	43
Table 3-19. Function adc_inserted_channel_config .....	44
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 .....	51
Table 3-24. Function adc_regular_data_read .....	51
Table 3-25. Function adc_inserted_data_read .....	52
Table 3-26. Function adc_sync_mode_convert_value_read .....	53
Table 3-27. Function adc_flag_get .....	54
Table 3-28. Function adc_flag_clear .....	54
Table 3-29. Function adc_interrupt_flag_get .....	55
Table 3-30. Function adc_interrupt_flag_clear .....	56
Table 3-31. Function adc_interrupt_enable .....	57
Table 3-32. Function adc_interrupt_disable .....	58
Table 3-33. Function adc_watchdog_single_channel_enable .....	59
Table 3-34. Function adc_watchdog_group_channel_enable .....	60
Table 3-35. Function adc_watchdog_disable .....	60
Table 3-36. Function adc_watchdog_threshold_config .....	61
Table 3-37. Function adc_oversample_mode_config .....	62
Table 3-38. Function adc_oversample_mode_enable .....	64

Table 3-39. Function adc_oversample_mode_disable .....	65
Table 3-40. BKP Registers .....	66
Table 3-41. BKP firmware function.....	66
Table 3-42. Function bkp_deinit.....	67
Table 3-43. Function bkp_write_data .....	67
Table 3-44. Function bkp_data_read .....	68
Table 3-45. Function bkp_rtc_calibration_output_enable.....	69
Table 3-46. Function bkp_rtc_calibration_output_disable.....	69
Table 3-47. Function bkp_rtc_signal_output_enable.....	70
Table 3-48. Function bkp_rtc_signal_output_disable.....	71
Table 3-49. Function bkp_rtc_output_select .....	71
Table 3-50. Function bkp_rtc_clock_output_select .....	72
Table 3-51. Function bkp_rtc_clock_calibration_direction.....	73
Table 3-52. Function bkp_rtc_calibration_value_set .....	74
Table 3-53. Function bkp_tamper_detection_enable .....	74
Table 3-54. Function bkp_tamper_detection_disable .....	75
Table 3-55. Function bkp_tamper_active_level_set .....	75
Table 3-56. Function bkp_tamper_interrupt_enable .....	76
Table 3-57. Function bkp_tamper_interrupt_disable .....	77
Table 3-58. Function bkp_flag_get .....	77
Table 3-59. Function bkp_flag_clear .....	78
Table 3-60. Function bkp_interrupt_flag_get .....	79
Table 3-61. Function bkp_interrupt_flag_clear .....	79
Table 3-62. CAN Registers .....	80
Table 3-63. CAN firmware function .....	81
Table 3-64. Structure can_parameter_struct .....	82
Table 3-65. Structure can_transmit_message_struct .....	83
Table 3-66. Structure can_receive_message_struct .....	83
Table 3-67. Structure can_filter_parameter_struct.....	84
Table 3-68. Function can_deinit.....	84
Table 3-69. Function can_struct_para_init .....	85
Table 3-70. Function can_init .....	86
Table 3-71. Function can_filter_init.....	87
Table 3-72. Function can1_filter_start_bank .....	88
Table 3-73. Function can_debug_freeze_enable .....	89
Table 3-74. Function can_debug_freeze_disable .....	90
Table 3-75. Function can_time_trigger_mode_enable.....	91
Table 3-76. Function can_time_trigger_mode_disable.....	92
Table 3-77. Function can_message_transmit.....	93
Table 3-78. Function can_transmit_states.....	95
Table 3-79. Function can_transmission_stop .....	96
Table 3-80. Function can_message_receive .....	97
Table 3-81. Function can_fifo_release .....	99
Table 3-82. Function can_receive_message_length_get .....	100

Table 3-83. Function can_working_mode_set.....	102
Table 3-84. Function can_wakeup .....	102
Table 3-85. Function can_error_get .....	103
Table 3-86. Function can_receive_error_number_get .....	105
Table 3-87. Function can_transmit_error_number_get .....	106
Table 3-88. Function can_interrupt_enable .....	107
Table 3-89. Function can_interrupt_disable .....	109
Table 3-90. Function can_flag_get .....	110
Table 3-91. Function can_flag_clear .....	111
Table 3-92. Function can_interrupt_flag_get.....	112
Table 3-93. Function can_interrupt_flag_clear .....	114
Table 3-94. CRC Registers .....	115
Table 3-95. CRC firmware function .....	115
Table 3-96. Function crc_deinit.....	115
Table 3-97. Function crc_data_register_reset.....	116
Table 3-98. Function crc_data_register_read.....	117
Table 3-99. Function crc_free_data_register_read.....	117
Table 3-100. Function crc_free_data_register_write.....	118
Table 3-101. Function crc_single_data_calculate .....	119
Table 3-102. Function crc_block_data_calculate .....	119
Table 3-103. CTC Registers .....	120
Table 3-104. CTC firmware function.....	121
Table 3-105. Function ctc_deinit.....	122
Table 3-106. Function ctc_counter_enable .....	122
Table 3-107. Function ctc_counter_disable .....	123
Table 3-108. Function ctc_irc48m_trim_value_config .....	124
Table 3-109. Function ctc_software_refsource_pulse_generate .....	124
Table 3-110. Function ctc_hardware_trim_mode_config .....	125
Table 3-111. Function ctc_refsource_polarity_config.....	126
Table 3-112. Function ctc_refsource_signal_select.....	126
Table 3-113. Function ctc_refsource_prescaler_config .....	127
Table 3-114. Function ctc_clock_limit_value_config .....	128
Table 3-115. Function ctc_counter_reload_value_config .....	129
Table 3-116. Function ctc_counter_capture_value_read .....	129
Table 3-117. Function ctc_counter_direction_read .....	130
Table 3-118. Function ctc_counter_reload_value_read.....	131
Table 3-119. Function ctc_irc48m_trim_value_read.....	131
Table 3-120. Function ctc_interrupt_enable .....	132
Table 3-121. Function ctc_interrupt_disable .....	133
Table 3-122. Function ctc_interrupt_flag_get.....	133
Table 3-123. Function ctc_interrupt_flag_clear .....	134
Table 3-124. Function ctc_flag_get .....	135
Table 3-125. Function ctc_flag_clear .....	136
Table 3-126. DAC Registers .....	错误!未定义书签。

Table 3-127. DAC firmware function .....	错误!未定义书签。
Table 3-128. Function dac_deinit.....	错误!未定义书签。
Table 3-129. Function dac_enable.....	错误!未定义书签。
Table 3-130. Function dac_disable.....	错误!未定义书签。
Table 3-131. Function dac_dma_enable.....	错误!未定义书签。
Table 3-132. Function dac_dma_disable.....	错误!未定义书签。
Table 3-133. Function dac_output_buffer_enable .....	错误!未定义书签。
Table 3-134. Function dac_output_buffer_disable .....	错误!未定义书签。
Table 3-135. Function dac_output_value_get .....	错误!未定义书签。
Table 3-136. Function dac_data_set .....	错误!未定义书签。
Table 3-137. Function dac_trigger_enable.....	错误!未定义书签。
Table 3-138. Function dac_trigger_disable.....	错误!未定义书签。
Table 3-139. Function dac_trigger_source_config.....	错误!未定义书签。
Table 3-140. Function dac_software_trigger_enable .....	错误!未定义书签。
Table 3-141. Function dac_software_trigger_disable .....	错误!未定义书签。
Table 3-142. Function dac_wave_mode_config.....	错误!未定义书签。
Table 3-143. Function dac_wave_bit_width_config .....	错误!未定义书签。
Table 3-144. Function dac_lfsr_noise_config .....	错误!未定义书签。
Table 3-145. Function dac_triangle_noise_config.....	错误!未定义书签。
Table 3-146. Function dac_concurrent_enable.....	错误!未定义书签。
Table 3-147. Function dac_concurrent_disable.....	错误!未定义书签。
Table 3-148. Function dac_concurrent_software_trigger_enable .....	错误!未定义书签。
Table 3-149. Function dac_concurrent_software_trigger_disable .....	错误!未定义书签。
Table 3-150. Function dac_concurrent_output_buffer_enable .....	错误!未定义书签。
Table 3-151. Function dac_concurrent_output_buffer_disable .....	错误!未定义书签。
Table 3-152. Function dac_concurrent_data_set .....	错误!未定义书签。
Table 3-153. DBG Registers.....	152
Table 3-154. DBG firmware function .....	152
Table 3-155. Enum dbg_periph_enum .....	153
Table 3-156. Function dbg_deinit .....	154
Table 3-157. Function dbg_id_get .....	154
Table 3-158. Function dbg_low_power_enable.....	155
Table 3-159. Function dbg_low_power_disable.....	156
Table 3-160. Function dbg_periph_enable .....	156
Table 3-161. Function dbg_periph_disable.....	157
Table 3-162. Function dbg_trace_pin_enable .....	158
Table 3-163. Function dbg_trace_pin_disable .....	159
Table 3-164. DMA Registers.....	159
Table 3-165. DMA firmware function .....	160
Table 3-166. Structure dma_parameter_struct.....	161
Table 3-167. Function dma_deinit .....	162
Table 3-168. Function dma_struct_para_init .....	162
Table 3-169. Function dma_init.....	163
Table 3-170. Function dma_circulation_enable .....	164

Table 3-171. Function dma_circulation_disable .....	165
Table 3-172. Function dma_memory_to_memory_enable .....	166
Table 3-173. Function dma_memory_to_memory_disable .....	166
Table 3-174. Function dma_channel_enable .....	167
Table 3-175. Function dma_channel_disable .....	168
Table 3-176. Function dma_periph_address_config .....	169
Table 3-177. Function dma_memory_address_config .....	170
Table 3-178. Function dma_transfer_number_config .....	171
Table 3-179. Function dma_transfer_number_get .....	172
Table 3-180. Function dma_priority_config .....	172
Table 3-181. Function dma_memory_width_config .....	173
Table 3-182. Function dma_periph_width_config .....	174
Table 3-183. Function dma_memory_increase_enable .....	175
Table 3-184. Function dma_memory_increase_disable .....	176
Table 3-185. Function dma_periph_increase_enable .....	177
Table 3-186. Function dma_periph_increase_disable .....	178
Table 3-187. Function dma_transfer_direction_config .....	178
Table 3-188. Function dma_flag_get .....	179
Table 3-189. Function dma_flag_clear .....	180
Table 3-190. Function dma_interrupt_flag_get .....	181
Table 3-191. Function dma_interrupt_flag_clear .....	182
Table 3-192. Function dma_interrupt_enable .....	184
Table 3-193. Function dma_interrupt_disable .....	185
Table 3-194. EXMC Registers .....	186
Table 3-195. EXMC firmware function .....	186
Table 3-196. Structure exmc_norsram_timing_parameter_struct .....	187
Table 3-197. Structure exmc_norsram_parameter_struct .....	188
Table 3-198. Structure exmc_nand_pccard_timing_parameter_struct .....	188
Table 3-199. Structure exmc_nand_parameter_struct .....	189
Table 3-200. Structure exmc_pccard_parameter_struct .....	189
Table 3-201. Function exmc_norsram_deinit .....	190
Table 3-202. Function exmc_norsram_struct_para_init .....	190
Table 3-203. Function exmc_norsram_init .....	191
Table 3-204. Function exmc_norsram_enable .....	192
Table 3-205. Function exmc_norsram_disable .....	193
Table 3-206. Function exmc_nand_deinit .....	194
Table 3-207. Function exmc_nand_struct_para_init .....	194
Table 3-208. Function exmc_nand_init .....	195
Table 3-209. Function exmc_nand_enable .....	196
Table 3-210. Function exmc_nand_disable .....	197
Table 3-211. Function exmc_pccard_deinit .....	198
Table 3-212. Function exmc_pccard_struct_para_init .....	198
Table 3-213. Function exmc_pccard_init .....	199
Table 3-214. Function exmc_pccard_enable .....	200

Table 3-215. Function <code>exmc_pccard_disable</code> .....	201
Table 3-216. Function <code>exmc_norsram_page_size_config</code> .....	201
Table 3-217. Function <code>exmc_nand_ecc_config</code> .....	202
Table 3-218. Function <code>exmc_ecc_get</code> .....	203
Table 3-219. Function <code>exmc_interrupt_enable</code> .....	204
Table 3-220. Function <code>exmc_interrupt_disable</code> .....	205
Table 3-221. Function <code>exmc_flag_get</code> .....	206
Table 3-222. Function <code>exmc_flag_clear</code> .....	207
Table 3-223. Function <code>exmc_interrupt_flag_get</code> .....	208
Table 3-224. Function <code>exmc_interrupt_flag_clear</code> .....	209
Table 3-225. EXTI Registers .....	210
Table 3-226. EXTI firmware function .....	210
Table 3-227. Function <code>exti_deinit</code> .....	212
Table 3-228. Function <code>exti_init</code> .....	212
Table 3-229. Function <code>exti_interrupt_enable</code> .....	213
Table 3-230. Function <code>exti_interrupt_disable</code> .....	213
Table 3-231. Function <code>exti_event_enable</code> .....	214
Table 3-232. Function <code>exti_event_disable</code> .....	214
Table 3-233. Function <code>exti_software_interrupt_enable</code> .....	215
Table 3-234. Function <code>exti_software_interrupt_disable</code> .....	215
Table 3-235. Function <code>exti_flag_get</code> .....	216
Table 3-236. Function <code>exti_flag_clear</code> .....	216
Table 3-237. Function <code>exti_interrupt_flag_get</code> .....	217
Table 3-238. Function <code>exti_interrupt_flag_clear</code> .....	217
Table 3-239. FMC Registers .....	218
Table 3-240. FMC firmware function .....	218
Table 3-241. Enum <code>fmc_state_enum</code> .....	220
Table 3-242. Enum <code>fmc_int_enum</code> .....	220
Table 3-243. Enum <code>fmc_flag_enum</code> .....	221
Table 3-244. Enum <code>fmc_interrupt_flag_enum</code> .....	221
Table 3-245. Function <code>fmc_wscnt_set</code> .....	222
Table 3-246. Function <code>fmc_unlock</code> .....	222
Table 3-247. Function <code>fmc_bank0_unlock</code> .....	223
Table 3-248. Function <code>fmc_bank1_unlock</code> .....	224
Table 3-249. Function <code>fmc_lock</code> .....	224
Table 3-250. Function <code>fmc_bank0_lock</code> .....	225
Table 3-251. Function <code>fmc_bank1_lock</code> .....	226
Table 3-252. Function <code>fmc_page_erase</code> .....	226
Table 3-253. Function <code>fmc_mass_erase</code> .....	227
Table 3-254. Function <code>fmc_bank0_erase</code> .....	227
Table 3-255. Function <code>fmc_bank1_erase</code> .....	228
Table 3-256. Function <code>fmc_word_program</code> .....	229
Table 3-257. Function <code>fmc_halfword_program</code> .....	229
Table 3-258. Function <code>fmc_word_reprogram</code> .....	230

Table 3-259. Function ob_unlock.....	231
Table 3-260. Function ob_lock .....	232
Table 3-261. Function ob_erase .....	232
Table 3-262. Function ob_write_protection_enable .....	233
Table 3-263. Function ob_security_protection_config .....	234
Table 3-264. Function ob_user_write .....	234
Table 3-265. Function ob_data_program.....	236
Table 3-266. Function ob_user_get .....	236
Table 3-267. Function ob_data_program.....	237
Table 3-268. Function ob_write_protection_get.....	238
Table 3-269. Function ob_spc_get.....	238
Table 3-270. Function fmc_interrupt_enable .....	239
Table 3-271. Function fmc_interrupt_disable .....	240
Table 3-272. Function fmc_flag_get .....	241
Table 3-273. Function fmc_flag_clear .....	242
Table 3-274. Function fmc_interrupt_flag_get .....	243
Table 3-275. Function fmc_interrupt_flag_clear .....	244
Table 3-276. Function fmc_bank0_state_get .....	245
Table 3-277. Function fmc_bank1_state_get .....	245
Table 3-278. Function fmc_bank0_ready_wait .....	246
Table 3-279. Function fmc_bank1_ready_wait .....	247
Table 3-280. FWDGT Registers .....	247
Table 3-281. FWDGT firmware function.....	248
Table 3-282. Function fwdgt_write_ensable .....	248
Table 3-283. Function fwdgt_write_disable .....	249
Table 3-284. Function fwdgt_enable .....	249
Table 3-285. Function fwdgt_counter_reload.....	250
Table 3-286. Function fwdgt_config.....	250
Table 3-287. Function fwdgt_flag_get fwdgt_write_disable .....	251
Table 3-288. GPIO Registers.....	253
Table 3-289. GPIO firmware function .....	253
Table 3-290. Function gpio_deinit .....	254
Table 3-291. Function gpio_afio_deinit .....	255
Table 3-292. Function gpio_init.....	255
Table 3-293. Function gpio_bit_set .....	257
Table 3-294. Function gpio_bit_reset .....	258
Table 3-295. Function gpio_bit_write.....	258
Table 3-296. Function gpio_port_write .....	259
Table 3-297. Function gpio_input_bit_get.....	260
Table 3-298. Function gpio_input_port_get.....	261
Table 3-299. Function gpio_output_bit_get .....	262
Table 3-300. Function gpio_output_port_get .....	262
Table 3-301. Function gpio_pin_remap_config.....	263
Table 3-302. Function gpio_exti_source_select .....	266



Table 3-303. Function gpio_event_output_config .....	266
Table 3-304. Function gpio_event_output_enable .....	267
Table 3-305. Function gpio_event_output_disable .....	268
Table 3-306. Function gpio_pin_lock .....	268
Table 3-307. Function gpio_compensation_config .....	269
Table 3-308. Function gpio_compensation_flag_get .....	270
Table 3-309. I2C Registers .....	271
Table 3-310. I2C firmware function.....	271
Table 3-311. Function i2c_deinit .....	273
Table 3-312. Function i2c_clock_config.....	273
Table 3-313. Function i2c_mode_addr_config .....	274
Table 3-314. Function i2c_smbus_type_config .....	275
Table 3-315. Function i2c_ack_config .....	276
Table 3-316. Function i2c_ackpos_config .....	277
Table 3-317. Function i2c_master_addressing .....	278
Table 3-318. Function i2c_dualaddr_enable .....	279
Table 3-319. Function i2c_enable .....	279
Table 3-320. Function i2c_disable .....	280
Table 3-321. Function i2c_start_on_bus .....	281
Table 3-322. Function i2c_stop_on_bus .....	281
Table 3-323. Function i2c_data_transmit .....	282
Table 3-324. Function i2c_data_receive .....	283
Table 3-325. Function i2c_dma_config .....	283
Table 3-326. Function i2c_dma_last_transfer_config .....	284
Table 3-327. Function i2c_stretch_scl_low_config .....	285
Table 3-328. Function i2c_slave_response_to_gcall_config.....	286
Table 3-329. Function i2c_software_reset_config .....	287
Table 3-330. Function i2c_pec_config .....	287
Table 3-331. Function i2c_pec_transfer_config .....	288
Table 3-332. Function i2c_pec_value_get .....	289
Table 3-333. Function i2c_smbus_alert_config .....	290
Table 3-334. Function i2c_smbus_arb_config .....	291
Table 3-335. Function i2c_flag_get .....	291
Table 3-336. Function i2c_flag_clear .....	293
Table 3-337. Function i2c_interrupt_enable .....	294
Table 3-338. Function i2c_interrupt_disable .....	295
Table 3-339. Function i2c_interrupt_flag_get.....	296
Table 3-340. Function i2c_interrupt_flag_clear.....	298
Table 3-341. NVIC Registers .....	299
Table 3-342. SysTick Registers .....	300
Table 3-343. IRQn_Type .....	300
Table 3-344. MISC firmware function .....	302
Table 3-345. Function nvic_priority_group_set .....	302
Table 3-346. Function nvic_irq_enable .....	303

Table 3-347. Function nvic_irq_disable.....	304
Table 3-348. Function nvic_vector_table_set.....	304
Table 3-349. Function system_lowpower_set .....	305
Table 3-350. Function system_lowpower_reset.....	305
Table 3-351. Function systick_clksource_set .....	306
Table 3-352. PMU Registers.....	307
Table 3-353. PMU firmware function .....	307
Table 3-354. Function pmu_deinit .....	308
Table 3-355. Function pmu_lvd_select.....	308
Table 3-356. Function pmu_ldo_output_select.....	309
Table 3-357. Function pmu_highdriver_switch_select .....	310
Table 3-358. Function pmu_highdriver_mode_enable .....	311
Table 3-359. Function pmu_highdriver_mode_disable .....	311
Table 3-360. Function pmu_lowdriver_mode_enable .....	312
Table 3-361. Function pmu_lowdriver_mode_disable .....	313
Table 3-362. Function pmu_lvd_disable.....	313
Table 3-363. Function pmu_lowpower_driver_config.....	314
Table 3-364. Function pmu_normalpower_driver_config .....	315
Table 3-365. Function pmu_to_sleepmode.....	315
Table 3-366. Function pmu_to_deepsleepmode .....	316
Table 3-367. Function pmu_to_standbymode .....	317
Table 3-368. Function pmu_backup_write_enable .....	317
Table 3-369. Function pmu_backup_write_disable .....	318
Table 3-370. Function pmu_wakeup_pin_enable .....	319
Table 3-371. Function pmu_wakeup_pin_disable .....	319
Table 3-372. Function pmu_flag_get.....	320
Table 3-373. Function pmu_flag_clear.....	321
Table 3-374. RCU Registers .....	322
Table 3-375. RCU firmware function .....	322
Table 3-376. Enum rcu_periph_enum .....	324
Table 3-377. Enum rcu_periph_sleep_enum .....	325
Table 3-378. Enum rcu_periph_reset_enum.....	326
Table 3-379. Enum rcu_flag_enum.....	327
Table 3-380. Enum rcu_int_flag_enum .....	327
Table 3-381. Enum rcu_int_flag_clear_enum .....	328
Table 3-382. Enum rcu_int_enum.....	328
Table 3-383. Enum rcu_osci_type_enum .....	328
Table 3-384. Enum rcu_clock_freq_enum.....	329
Table 3-385. Function rcu_deinit .....	329
Table 3-386. Function rcu_periph_clock_enable .....	329
Table 3-387. Function rcu_periph_clock_disable .....	330
Table 3-388. Function rcu_periph_clock_sleep_enable .....	331
Table 3-389. Function rcu_periph_clock_sleep_disable .....	331
Table 3-390. Function rcu_periph_reset_enable.....	332

Table 3-391. Function rcu_periph_reset_disable .....	333
Table 3-392. Function rcu_bkp_reset_enable .....	333
Table 3-393. Function rcu_bkp_reset_disable .....	334
Table 3-394. Function rcu_system_clock_source_config.....	335
Table 3-395. Function rcu_system_clock_source_get .....	335
Table 3-396. Function rcu_ahb_clock_config .....	336
Table 3-397. Function rcu_apb1_clock_config .....	337
Table 3-398. Function rcu_apb2_clock_config .....	338
Table 3-399. Function rcu_ckout0_config.....	338
Table 3-400. Function rcu_pll_config .....	340
Table 3-401. Function rcu_pllpre sel_config .....	340
Table 3-402. Function rcu_predv0_config .....	341
Table 3-403. Function rcu_predv1_config .....	342
Table 3-404. Function rcu_pll1_config .....	343
Table 3-405. Function rcu_pll2_config .....	343
Table 3-406. Function rcu_adc_clock_config.....	344
Table 3-407. Function rcu_usb_clock_config .....	345
Table 3-408. Function rcu_rtc_clock_config .....	346
Table 3-409. Function rcu_i2s1_clock_config.....	347
Table 3-410. Function rcu_i2s2_clock_config.....	348
Table 3-411. Function rcu_ck48m_clock_config.....	348
Table 3-412. Function rcu_flag_get .....	349
Table 3-413. Function rcu_all_reset_flag_clear .....	350
Table 3-414. Function rcu_interrupt_flag_get .....	350
Table 3-415. Function rcu_interrupt_flag_clear .....	351
Table 3-416. Function rcu_interrupt_enable.....	352
Table 3-417. Function rcu_interrupt_disable.....	352
Table 3-418. Function rcu_lxtal_drive_capability_config.....	353
Table 3-419. Function rcu_osci_stab_wait .....	354
Table 3-420. Function rcu_osci_on .....	354
Table 3-421. Function rcu_osci_off .....	355
Table 3-422. Function rcu_osci_bypass_mode_enable .....	356
Table 3-423. Function rcu_osci_bypass_mode_disable .....	356
Table 3-424. Function rcu_hxtal_clock_monitor_enable .....	357
Table 3-425. Function rcu_hxtal_clock_monitor_disable .....	358
Table 3-426. Function rcu_irc8m_adjust_value_set.....	358
Table 3-427. Function rcu_deepsleep_voltage_set.....	359
Table 3-428. Function rcu_clock_freq_get.....	360
Table 3-429. RTC Registers .....	361
Table 3-430. RTC firmware function.....	361
Table 3-431. Function rtc_interrupt_enable .....	362
Table 3-432. Function rtc_interrupt_disable.....	363
Table 3-433. Function rtc_configuration_mode_enter.....	363
Table 3-434. Function rtc_configuration_mode_exit .....	364

Table 3-435. Function rtc_lwoff_wait .....	365
Table 3-436. Function rtc_register_sync_wait .....	365
Table 3-437. Function rtc_counter_get .....	366
Table 3-438. Function rtc_counter_set .....	366
Table 3-439. Function rtc_prescaler_set .....	367
Table 3-440. Function rtc_alarm_config .....	368
Table 3-441. Function rtc_divider_get .....	369
Table 3-442. Function rtc_flag_get .....	369
Table 3-443. Function rtc_flag_clear .....	370
Table 3-444. SDIO Registers .....	371
Table 3-445. SDIO firmware function .....	372
Table 3-446. Function sdio_deinit .....	374
Table 3-447. Function sdio_clock_config .....	374
Table 3-448. Function sdio_hardware_clock_enable .....	376
Table 3-449. Function sdio_hardware_clock_disable .....	376
Table 3-450. Function sdio_bus_mode_set .....	377
Table 3-451. Function sdio_power_state_set .....	378
Table 3-452. Function sdio_power_state_get .....	378
Table 3-453. Function sdio_clock_enable .....	379
Table 3-454. Function sdio_clock_disable .....	380
Table 3-455. Function sdio_command_response_config .....	380
Table 3-456. Function sdio_wait_type_set .....	381
Table 3-457. Function sdio_csm_enable .....	382
Table 3-458. Function sdio_csm_disable .....	383
Table 3-459. Function sdio_command_index_get .....	383
Table 3-460. Function sdio_response_get .....	384
Table 3-461. Function sdio_data_config .....	385
Table 3-462. Function sdio_data_transfer_config .....	386
Table 3-463. Function sdio_dsm_enable .....	387
Table 3-464. Function sdio_dsm_disable .....	388
Table 3-465. Function sdio_data_write .....	388
Table 3-466. Function sdio_data_read .....	389
Table 3-467. Function sdio_data_counter_get .....	390
Table 3-468. Function sdio_data_counter_get .....	390
Table 3-469. Function sdio_dma_enable .....	391
Table 3-470. Function sdio_dma_disable .....	392
Table 3-471. Function sdio_flag_get .....	392
Table 3-472. Function sdio_flag_clear .....	394
Table 3-473. Function sdio_interrupt_enable .....	395
Table 3-474. Function sdio_interrupt_disable .....	397
Table 3-475. Function sdio_interrupt_flag_get .....	398
Table 3-476. Function sdio_interrupt_flag_clear .....	400
Table 3-477. Function sdio_readwait_enable .....	402
Table 3-478. Function sdio_readwait_disable .....	402

Table 3-479. Function <code>sdio_stop_readwait_enable</code> .....	403
Table 3-480. Function <code>sdio_stop_readwait_disable</code> .....	403
Table 3-481. Function <code>sdio_readwait_type_set</code> .....	404
Table 3-482. Function <code>sdio_operation_enable</code> .....	405
Table 3-483. Function <code>sdio_operation_disable</code> .....	405
Table 3-484. Function <code>sdio_suspend_enable</code> .....	406
Table 3-485. Function <code>sdio_suspend_disable</code> .....	407
Table 3-486. Function <code>sdio_ceata_command_enable</code> .....	407
Table 3-487. Function <code>sdio_ceata_command_disable</code> .....	408
Table 3-488. Function <code>sdio_ceata_interrupt_enable</code> .....	408
Table 3-489. Function <code>sdio_ceata_interrupt_disable</code> .....	409
Table 3-490. Function <code>sdio_ceata_command_completion_enable</code> .....	410
Table 3-491. Function <code>sdio_ceata_command_completion_disable</code> .....	410
Table 3-492. SPI/I2S Registers .....	411
Table 3-493. SPI/I2S firmware function .....	412
Table 3-494. <code>spi_parameter_struct</code> .....	413
Table 3-495. Function <code>spi_i2s_deinit</code> .....	414
Table 3-496. Function <code>spi_struct_para_init</code> .....	414
Table 3-497. Function <code>spi_init</code> .....	415
Table 3-498. Function <code>spi_enable</code> .....	416
Table 3-499. Function <code>spi_disable</code> .....	417
Table 3-500. Function <code>i2s_init</code> .....	417
Table 3-501. Function <code>i2s_psc_config</code> .....	419
Table 3-502. Function <code>i2s_enable</code> .....	421
Table 3-503. Function <code>i2s_disable</code> .....	421
Table 3-504. Function <code>spi_nss_output_enable</code> .....	422
Table 3-505. Function <code>spi_nss_output_disable</code> .....	423
Table 3-506. Function <code>spi_nss_internal_high</code> .....	423
Table 3-507. Function <code>spi_nss_internal_low</code> .....	424
Table 3-508. Function <code>spi_dma_enable</code> .....	424
Table 3-509. Function <code>spi_dma_disable</code> .....	425
Table 3-510. Function <code>spi_i2s_data_frame_format_config</code> .....	426
Table 3-511. Function <code>spi_i2s_data_transmit</code> .....	427
Table 3-512. Function <code>spi_i2s_data_receive</code> .....	428
Table 3-513. Function <code>spi_bidirectional_transfer_config</code> .....	428
Table 3-514. Function <code>spi_crc_polynomial_set</code> .....	429
Table 3-515. Function <code>spi_crc_polynomial_get</code> .....	430
Table 3-516. Function <code>spi_crc_on</code> .....	431
Table 3-517. Function <code>spi_crc_off</code> .....	431
Table 3-518. Function <code>spi_crc_next</code> .....	432
Table 3-519. Function <code>spi_crc_get</code> .....	433
Table 3-520. Function <code>spi_ti_mode_enable</code> .....	433
Table 3-521. Function <code>spi_ti_mode_disable</code> .....	434
Table 3-522. Function <code>spi_nssp_mode_enable</code> .....	435

Table 3-523. Function spi_nssp_mode_disable.....	435
Table 3-524. Function qspi_enable .....	436
Table 3-525. Function qspi_disable .....	437
Table 3-526. Function qspi_write_enable.....	437
Table 3-527. Function qspi_read_enable .....	438
Table 3-528. Function qspi_io23_output_enable .....	439
Table 3-529. Function qspi_io23_output_disable .....	439
Table 3-530. Function spi_i2s_interrupt_enable.....	440
Table 3-531. Function spi_i2s_interrupt_disable.....	441
Table 3-532. Function spi_i2s_interrupt_flag_get .....	442
Table 3-533. Function spi_i2s_flag_get .....	443
Table 3-534. Function spi_crc_error_clear .....	444
Table 3-535. TIMERx Registers .....	445
Table 3-536. TIMERx firmware function.....	446
Table 3-537. Structure timer_parameter_struct .....	449
Table 3-538. Structure timer_break_parameter_struct.....	449
Table 3-539. Structure timer_oc_parameter_struct.....	450
Table 3-540. Structure timer_ic_parameter_struct.....	450
Table 3-541. Function timer_deinit.....	451
Table 3-542. Function timer_struct_para_init.....	451
Table 3-543. Function timer_init .....	452
Table 3-544. Function timer_enable .....	453
Table 3-545. Function timer_disable .....	454
Table 3-546. Function timer_auto_reload_shadow_enable .....	454
Table 3-547. Function timer_auto_reload_shadow_disable .....	455
Table 3-548. Function timer_update_event_enable .....	456
Table 3-549. Function timer_update_event_disable .....	456
Table 3-550. Function timer_counter_alignment .....	457
Table 3-551. Function timer_counter_up_direction .....	458
Table 3-552. Function timer_counter_down_direction .....	459
Table 3-553. Function timer_prescaler_config.....	459
Table 3-554. Function timer_repetition_value_config .....	460
Table 3-555. Function timer_autoreload_value_config .....	461
Table 3-556. Function timer_counter_value_config.....	462
Table 3-557. Function timer_counter_read .....	462
Table 3-558. Function timer_prescaler_read .....	463
Table 3-559. Function timer_single_pulse_mode_config .....	464
Table 3-560. Function timer_update_source_config.....	465
Table 3-561. Function timer_dma_enable .....	466
Table 3-562. Function timer_dma_disable .....	467
Table 3-563. Function timer_channel_dma_request_source_select.....	468
Table 3-564. Function timer_dma_transfer_config.....	468
Table 3-565. Function timer_event_software_generate.....	471
Table 3-566. Function timer_break_struct_para_init .....	472

Table 3-567. Function timer_break_config .....	472
Table 3-568. Function timer_break_enable .....	473
Table 3-569. Function timer_break_disable .....	474
Table 3-570. Function timer_automatic_output_enable .....	475
Table 3-571. Function timer_automatic_output_disable .....	475
Table 3-572. Function timer_primary_output_config .....	476
Table 3-573. Function timer_channel_control_shadow_config .....	477
Table 3-574. Function timer_channel_control_shadow_update_config .....	478
Table 3-575. Function timer_channel_output_struct_para_init .....	479
Table 3-576. Function timer_channel_output_config .....	479
Table 3-577. Function timer_channel_output_mode_config .....	481
Table 3-578. Function timer_channel_output_pulse_value_config .....	482
Table 3-579. Function timer_channel_output_shadow_config .....	483
Table 3-580. Function timer_channel_output_fast_config .....	484
Table 3-581. Function timer_channel_output_clear_config .....	485
Table 3-582. Function timer_channel_output_polarity_config .....	486
Table 3-583. Function timer_channel_complementary_output_polarity_config .....	487
Table 3-584. Function timer_channel_output_state_config .....	488
Table 3-585. Function timer_channel_complementary_output_state_config .....	490
Table 3-586. Function timer_channel_input_struct_para_init .....	491
Table 3-587. Function timer_input_capture_config .....	491
Table 3-588. Function timer_channel_input_capture_prescaler_config .....	493
Table 3-589. Function timer_channel_capture_value_register_read .....	494
Table 3-590. Function timer_input_pwm_capture_config .....	495
Table 3-591. Function timer_hall_mode_config .....	496
Table 3-592. Function timer_input_trigger_source_select .....	496
Table 3-593. Function timer_master_output_trigger_source_select .....	498
Table 3-594. Function timer_slave_mode_select .....	499
Table 3-595. Function timer_master_slave_mode_config .....	500
Table 3-596. Function timer_external_trigger_config .....	501
Table 3-597. Function timer_quadrature_decoder_mode_config .....	502
Table 3-598. Function timer_internal_clock_config .....	504
Table 3-599. Function timer_internal_trigger_as_external_clock_config .....	504
Table 3-600. Function timer_external_trigger_as_external_clock_config .....	505
Table 3-601. Function timer_external_clock_mode0_config .....	506
Table 3-602. Function timer_external_clock_mode1_config .....	508
Table 3-603. Function timer_external_clock_mode1_disable .....	509
Table 3-604. Function timer_write_chxval_register_config .....	510
Table 3-605. Function timer_output_value_selection_config .....	510
Table 3-606. Function timer_interrupt_enable .....	511
Table 3-607. Function timer_interrupt_disable .....	512
Table 3-608. Function timer_interrupt_flag_get .....	513
Table 3-609. Function timer_interrupt_flag_clear .....	514
Table 3-610. Function timer_flag_get .....	515



Table 3-611. Function timer_flag_clear.....	517
Table 3-612. USART Registers .....	518
Table 3-613. USART firmware function.....	518
Table 3-614. Function usart_deinit.....	520
Table 3-615. Function usart_baudrate_set .....	521
Table 3-616. Function usart_parity_config .....	522
Table 3-617. Function usart_word_length_set.....	523
Table 3-618. Function usart_stop_bit_set.....	524
Table 3-619. Function usart_enable .....	525
Table 3-620. Function usart_disable .....	525
Table 3-621. Function usart_transmit_config.....	526
Table 3-622. Function usart_receive_config .....	527
Table 3-623. Function usart_data_first_config .....	528
Table 3-624. Function usart_invert_config .....	528
Table 3-625. Function usart_receiver_timeout_enable.....	530
Table 3-626. Function usart_receiver_timeout_disable.....	530
Table 3-627. Function usart_receiver_timeout_threshold_config.....	531
Table 3-628. Function usart_data_transmit .....	532
Table 3-629. Function usart_data_receive .....	532
Table 3-630. Function usart_address_config.....	533
Table 3-631. Function usart_mute_mode_enable.....	534
Table 3-632. Function usart_mute_mode_disable.....	535
Table 3-633. Function usart_mute_mode_wakeup_config .....	535
Table 3-634. Function usart_lin_mode_enable .....	536
Table 3-635. Function usart_lin_mode_disable .....	537
Table 3-636. Function usart_lin_break_dection_length_config.....	538
Table 3-637. Function usart_send_break.....	538
Table 3-638. Function usart_halfduplex_enable .....	539
Table 3-639. Function usart_halfduplex_disable .....	540
Table 3-640. Function usart_synchronous_clock_enable .....	540
Table 3-641. Function usart_synchronous_clock_disable .....	541
Table 3-642. Function usart_synchronous_clock_config.....	542
Table 3-643. Function usart_guard_time_config .....	543
Table 3-644. Function usart_smartcard_mode_enable .....	544
Table 3-645. Function usart_smartcard_mode_disable .....	544
Table 3-646. Function usart_smartcard_mode_nack_enable.....	545
Table 3-647. Function usart_smartcard_mode_nack_disable.....	545
Table 3-648. Function usart_smartcard_autoretry_config.....	546
Table 3-649. Function usart_block_length_config .....	547
Table 3-650. Function usart_irda_mode_enable.....	548
Table 3-651. Function usart_irda_mode_disable.....	548
Table 3-652. Function usart_prescaler_config.....	549
Table 3-653. Function usart_irda_lowpower_config .....	550
Table 3-654. Function usart_hardware_flow_rts_config .....	551





Table 3-655. Function <code>usart_hardware_flow_cts_config</code> .....	551
Table 3-656. Function <code>usart_dma_receive_config</code> .....	552
Table 3-657. Function <code>usart_dma_transmit_config</code> .....	553
Table 3-658. Function <code>usart_flag_get</code> .....	554
Table 3-659. Function <code>usart_flag_clear</code> .....	555
Table 3-660. Function <code>usart_interrupt_enable</code> .....	556
Table 3-661. Function <code>usart_interrupt_disable</code> .....	557
Table 3-662. Function <code>usart_interrupt_flag_get</code> .....	559
Table 3-663. Function <code>usart_interrupt_flag_clear</code> .....	560
Table 3-664. WWDGT Registers .....	561
Table 3-665. WWDGT firmware function .....	562
Table 3-666. Function <code>wwdgt_deinit</code> .....	562
Table 3-667. Function <code>wwdgt_enable</code> .....	563
Table 3-668. Function <code>wwdgt_counter_update</code> .....	563
Table 3-669. Function <code>wwdgt_config</code> .....	564
Table 3-670. Function <code>wwdgt_interrupt_enable</code> .....	565
Table 3-671. Function <code>wwdgt_flag_get</code> .....	565
Table 3-672. Function <code>wwdgt_flag_clear</code> .....	566
Table 4-1. Revision history .....	568

## 1. Introduction

This manual introduces firmware library of GD32F403xx devices which are 32-bit microcontrollers based on the ARM processor.

The firmware library is a firmware function package, including program, data structure and macro definitions, all the performance features of peripherals of GD32F403xx devices are involved in the package. The peripheral driving code and firmware examples on evaluation board are also included in firmware library. Users need not learn each peripherals in details and it's easy to apply a peripheral by using the firmware library. Using firmware library can greatly reduce programming time, thereby reducing development costs.

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

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

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

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

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

### 1.1. Rules of User Manual and Firmware Library

#### 1.1.1. Peripherals

Table 1-1. Peripherals

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

Peripherals	Descriptions
DBG	Debug
DMA	Direct memory access controller
EXMC	External memory controller
EXTI	Interrupt/event controller
FMC	Flash memory controller
FWDGT	Free watchdog timer
GPIO/AFIO	General-purpose and alternate-function I/Os
I2C	Inter-integrated circuit interface
MISC	Nested Vectored Interrupt Controller
PMU	Power management unit
RCU	Reset and clock unit
RTC	Real-time Clock
SDIO	SDIO interface
SPI/I2S	Serial peripheral interface/Inter-IC sound
TIMER	TIMER
USART	Universal synchronous/asynchronous receiver /transmitter
WWDGT	Window watchdog timer
USBFS	Universal serial bus full-speed interface

### 1.1.2. Naming rules

The firmware library naming rules are shown as below:

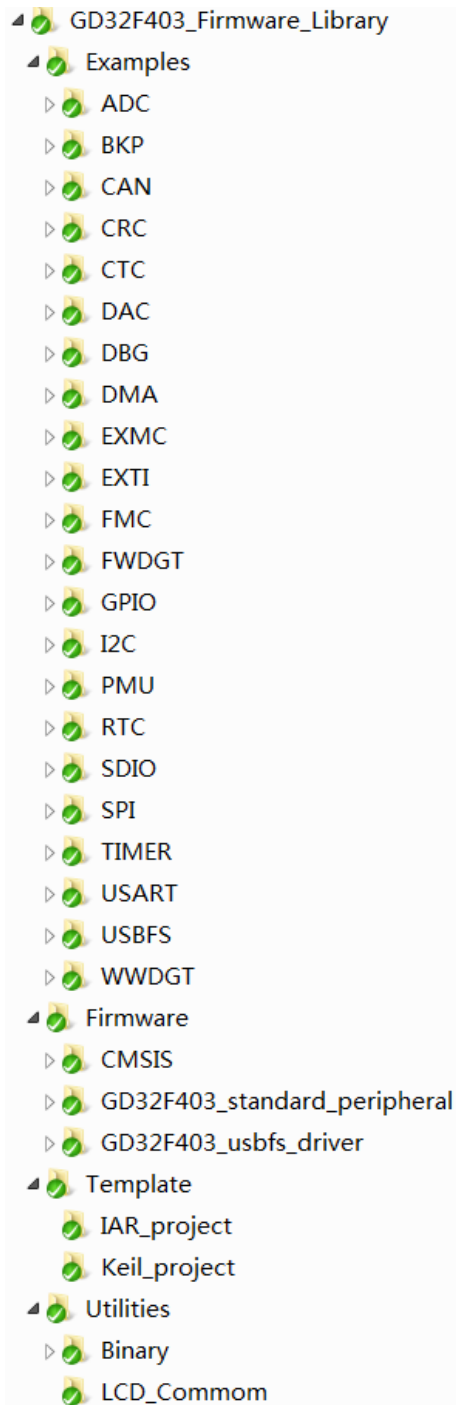
- The peripherals are shortened in XXX format, such as: ADC. More shorten information of peripherals refer to [Peripherals](#);
- The name of sourcefile and header file are started with “gd32f403\_”, such as: gd32f403\_adc.h;
- The constants used only in one file should be defined in the used file; the 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

GD32F403\_Firmware\_Library, the file structure is shown as below:

**Figure 2-1. File structure of firmware library of GD32F403**



### 2.1.1. Examples Folder

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

- readme.txt: the description and using guide of the example;
- gd32f403\_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);
- gd32f403\_it.c: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- gd32f403\_it.h: the header file include all the prototypes of the interrupt service routines;
- systick.c: the source file include the precise time delay functions by using systick;
- systick.h: the header file include the prototype of the precise time delay functions by using systick;
- main.c: example code. Note: all the examples are not influenced by software IDEs.

### 2.1.2. Firmware Folder

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

- CMSIS subfolder includes the Cortex M4 kernel support files, the startup file based on the Cortex M4 kernel processor, the global header file of GD32F403 and system configuration file;
- GD32F403\_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;
- GD32F403\_usbfs\_driver subfolder includes all the related files about USBFS peripheral:
  - Include subfolder includes the header files of USBFS peripheral, users need not modify this folder;
  - Source subfolder includes the source files of USBFS peripheral, users need not modify this folder;

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

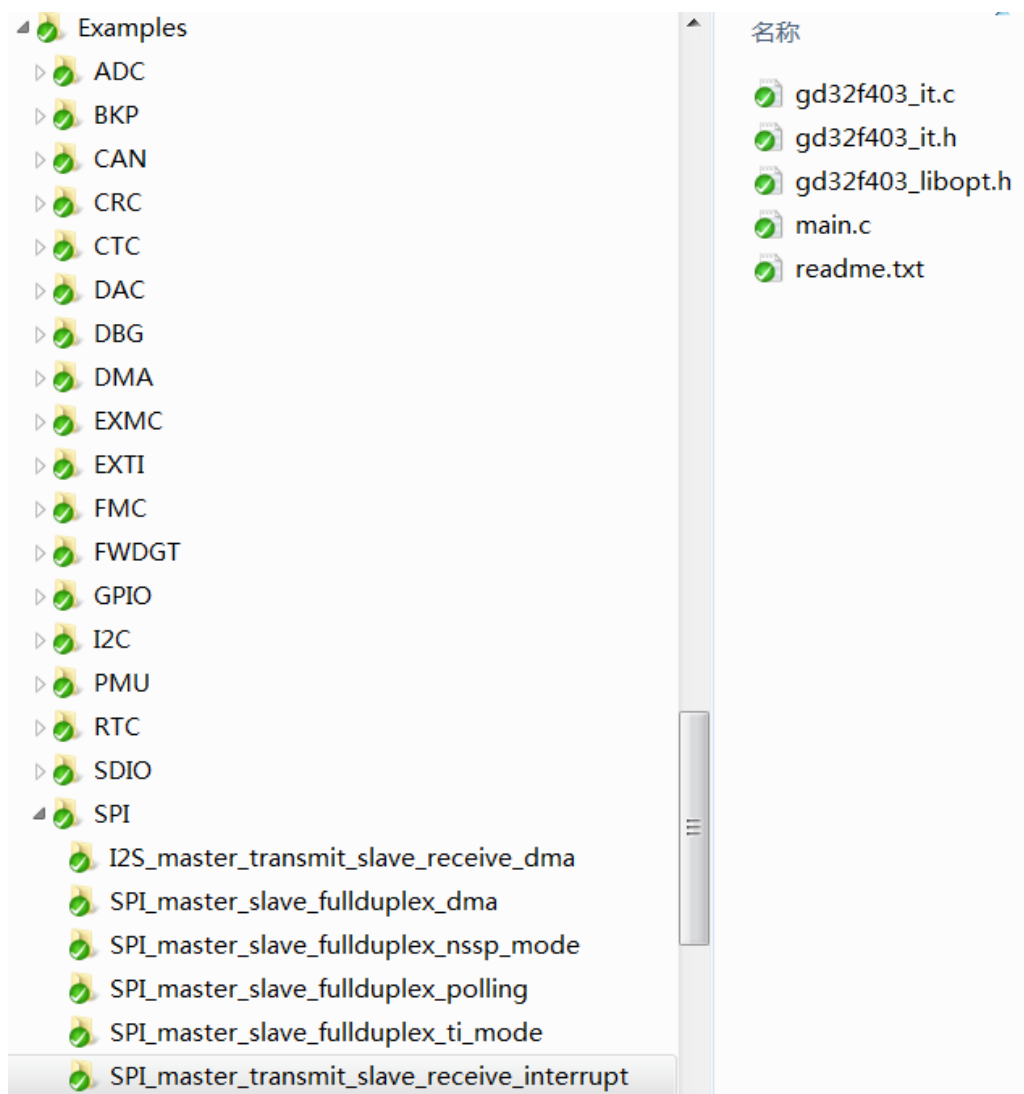
### 2.1.3. Template Folder

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

## Select files

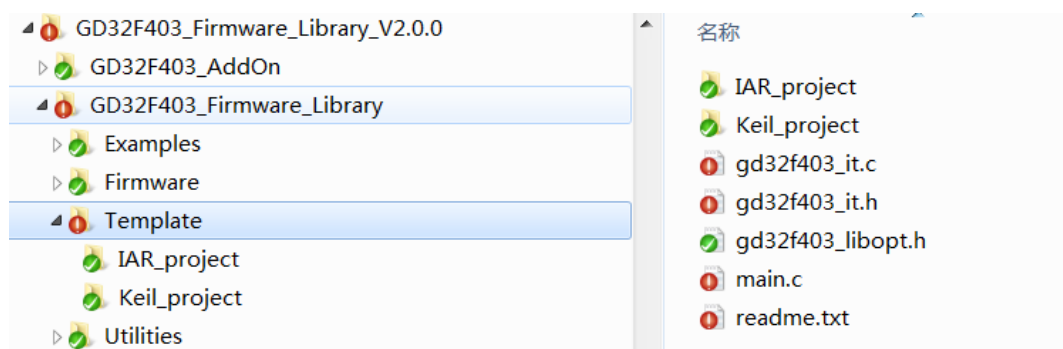
Open “Examples” folder, select the module to be tested, such as SPI, open “SPI” folder, select an example of SPI, such as “SPI\_master\_transmit\_slave\_receive\_interrupt”, shown as below:

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



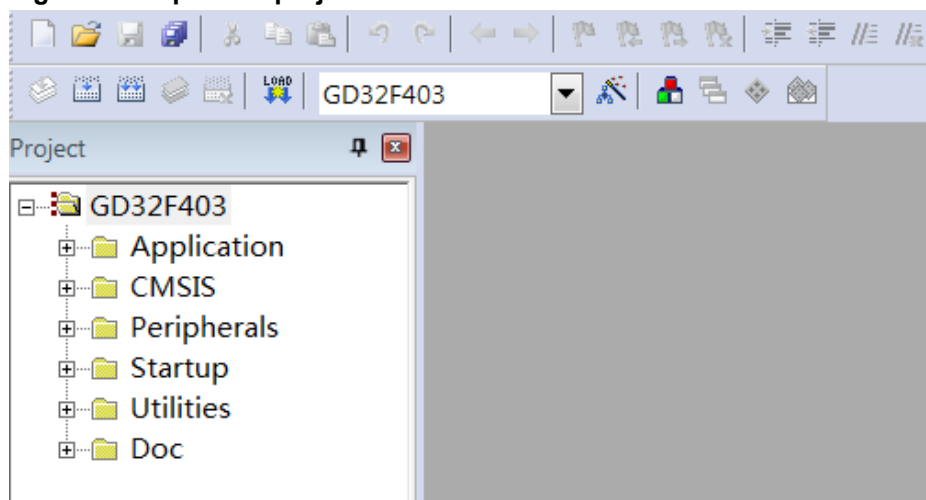
## Copy files

Open “Template” folder, keep the folders of “IAR\_project” and “Keil\_project”, and delete the other files, then copy all the files in “SPI\_master\_transmit\_slave\_receive\_interrupt” folder to the “Template” subfolder, shown as below:

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

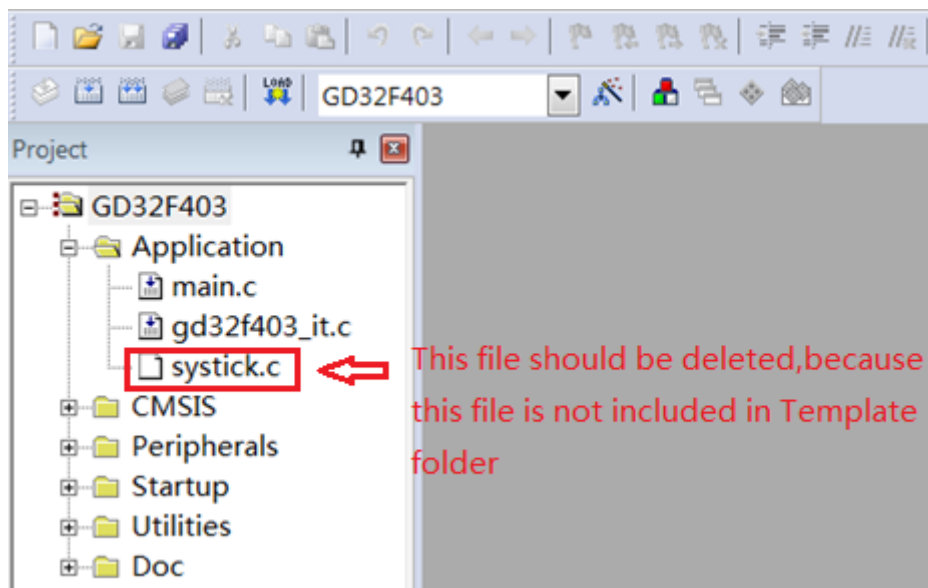
### Open a project

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

**Figure 2-4. Open the project file**

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

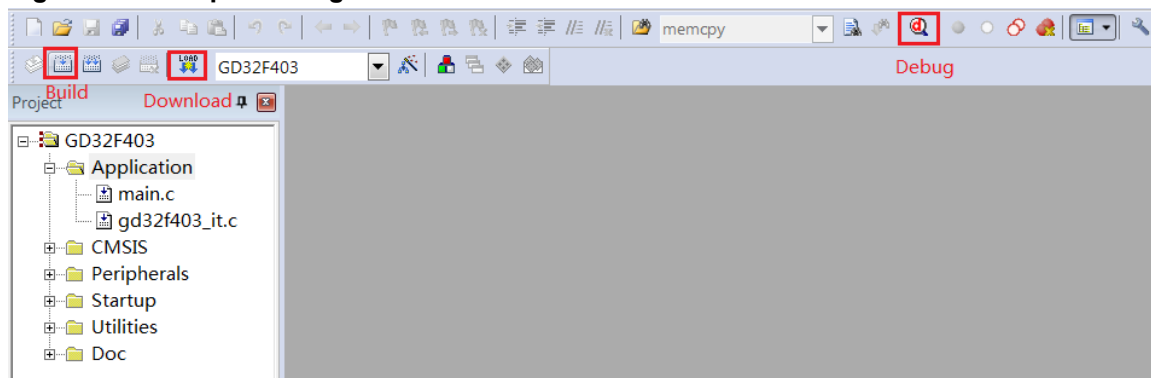
Figure 2-5. Configure project files



### Compile-Debug-Download

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

Figure 2-6. Compile-debug-download



#### 2.1.4. Utilities Folder

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

- Binary, LCD\_Commom include files for USB tests;
- gd32f403\_eval.h and gd32f403z\_lcd\_eval.h are related header files of the evaluation board about running the firmware examples;
- gd32f403\_eval.c and gd32f403z\_lcd\_eval.c are related source files of the evaluation board about running the firmware examples.



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

## 2.2. File descriptions of Firmware Library

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

**Table 2-1. Function descriptions of Firmware Library**

Files	Descriptions
gd32f403_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.
gd32f403_it.h	Header file, including all the prototypes of interrupt service routines.
gd32f403_it.c	Source files about interrupt service routines of peripherals. User can written his own interrupt functions in this file. For the different interrupt service requests to the same interrupt vector, users can confirm the interrupt source by functions of judging interrupt flags of peripherals. The functions are included in the firmware library.
gd32f403_xxx.h	The header file of peripheral PPP, including functions about peripheral PPP, and the variables used for functions.
gd32f403_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
ADC_WDLT	Watchdog low threshold register
ADC_RSQ0	Regular sequence register 0
ADC_RSQ1	Regular sequence register 1
ADC_RSQ2	Regular sequence register 2
ADC_ISQ	Inserted sequence register
ADC_IDATAx	Inserted data register x(x=0..3)
ADC_RDATA	Regular data register
ADC_OVSAMPCTL	Oversample control register

### 3.2.2. Descriptions of Peripheral functions

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

**Table 3-3. ADC firmware function**

Function name	Function description
adc_deinit	reset ADCx peripheral
adc_enable	enable ADC interface
adc_disable	disable ADC interface
adc_calibration_enable	ADC calibration and reset calibration
adc_dma_mode_enable	enable DMA request
adc_dma_mode_disable	disable DMA request
adc_tempsensor_vrefint_enable	enable the temperature sensor and Vrefint channel

Function name	Function description
adc_tempsensor_vrefint_disable	disable the temperature sensor and Vrefint channel
adc_resolution_config	configure ADC resolution
adc_discontinuous_mode_config	configure ADC discontinuous mode
adc_mode_config	configure the ADC sync mode
adc_special_function_config	enable or disable ADC special function
adc_data_alignment_config	configure ADC data alignment
adc_channel_length_config	configure the length of regular channel group or inserted channel group
adc_regular_channel_config	configure ADC regular channel
adc_inserted_channel_config	configure ADC inserted channel
adc_inserted_channel_offset_config	configure ADC inserted channel offset
adc_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
adc_regular_data_read	read ADC regular group data register
adc_inserted_data_read	read ADC inserted group data register
adc_sync_mode_convert_value_read	read the last ADC0 and ADC1 conversion result data in sync mode
adc_flag_get	get the ADC flag bits
adc_flag_clear	clear the ADC flag bits
adc_interrupt_flag_get	get the ADC interrupt bits
adc_interrupt_flag_clear	clear the ADC flag
adc_interrupt_enable	enable ADC interrupt
adc_interrupt_disable	disable ADC interrupt
adc_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

Function name	Function description
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\_deinit

The description of adc\_deinit is shown as below:

**Table 3-4. Function adc\_deinit**

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

Example:

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

## adc\_enable

The description of adc\_enable is shown as below:

**Table 3-5. Function adc\_enable**

Function name	adc_enable
Function prototype	void adc_enable(uint32_t adc_periph);

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

Example:

```
/* enable ADC0 */
```

```
adc_enable(ADC0);
```

### adc\_disable

The description of adc\_disable is shown as below:

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

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

Example:

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

### adc\_calibration\_enable

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

**Table 3-7. Function adc\_calibration\_enable**

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

Example:

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

### 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(uint32_t adc_periph);
<b>Function descriptions</b>	enable ADCx DMA request
<b>Precondition</b>	-

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

Example:

```
/* enable ADC0 DMA request */
adc_dma_mode_enable(ADC0);
```

### **adc\_dma\_mode\_disable**

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

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

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

Example:

```
/* disable ADC0 DMA request */
```



```
adc_dma_mode_disable(ADC0);
```

### adc\_tempsensor\_vrefint\_enable

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

**Table 3-10. Function adc\_tempsensor\_vrefint\_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 3-11. Function adc\_tempsensor\_vrefint\_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\_resolution\_config

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

**Table 3-12. Function adc\_resolution\_config**

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

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

Example:

```
/* configure ADC0 data alignment */
```

```
adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);
```

### adc\_discontinuous\_mode\_config

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

**Table 3-13. Function adc\_discontinuous\_mode\_config**

<b>Function name</b>	adc_discontinuous_mode_config
<b>Function prototype</b>	void adc_discontinuous_mode_config(uint32_t adc_periph, 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_periph</b>	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_channel_group</b>	select the channel group
ADC_REGULAR_CHANNEL	regular channel group
ADC_INSERTED_CHANNEL	inserted channel group
ADC_CHANNEL_DISCON_DISABLE	disable discontinuous mode of regular and inserted channel
<b>Input parameter{in}</b>	
<b>length</b>	number of conversions in discontinuous mode, the number can be 1..8 for regular channel, the number has no effect for inserted channel
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* configure ADC0 regular channel group discontinuous mode */
```

```
adc_discontinuous_mode_config(ADC0, ADC_REGULAR_CHANNEL, 6);
```

### adc\_mode\_config

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

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

<b>Function name</b>	adc_mode_config
<b>Function prototype</b>	void adc_mode_config(uint32_t mode);
<b>Function descriptions</b>	configure the ADCs sync mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	ADC mode
ADC_MODE_FREE	all the ADC work independently
ADC_DUAL_REGULAR_PARALLEL_INSERTED_PARALLEL	ADC0 and ADC1 work in combined regular parallel + inserted parallel mode
ADC_DUAL_REGULAR_PARALLEL_INSERTED_ROTATION	ADC0 and ADC1 work in combined regular parallel + trigger rotation mode
ADC_DUAL_INSERTED_PARALLEL_REGULAR_FOLLOWUP_FAST	ADC0 and ADC1 work in combined inserted parallel + follow-up fast mode
ADC_DUAL_INSERTED_PARALLEL_REGULAR_FOLLOWUP_SLOW	ADC0 and ADC1 work in combined inserted parallel + follow-up slow mode
ADC_DUAL_INSERTED	ADC0 and ADC1 work in inserted parallel mode only

<i>D_PARALLEL</i>	
<i>ADC_DAUL_REGULAL _PARALLEL</i>	ADC0 and ADC1 work in regular parallel mode only
<i>ADC_DAUL_REGULAL _FOLLOWUP_FAST</i>	ADC0 and ADC1 work in follow-up fast mode only
<i>ADC_DAUL_REGULAL _FOLLOWUP_SLOW</i>	ADC0 and ADC1 work in follow-up slow mode only
<i>ADC_DAUL_INSERTE D_TRIGGER_ROTAT ION</i>	ADC0 and ADC1 work in trigger rotation mode only
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the ADC sync mode */
adc_mode_config(ADC_MODE_FREE);
```

### adc\_special\_function\_config

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

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

<b>Function name</b>	adc_special_function_config
<b>Function prototype</b>	void adc_special_function_config(uint32_t adc_periph, 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>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<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 ADC0 scan mode */
```

```
adc_special_function_config(ADC0, ADC_SCAN_MODE, ENABLE);
```

### adc\_data\_alignment\_config

The description of `adc_data_alignment_config` is shown as below:

**Table 3-16. Function `adc_data_alignment_config`**

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

Input parameter{in}	
<b>data_alignment</b>	data alignment select
<i>ADC_DATAALIGN_RIGHT</i>	LSB alignment
<i>ADC_DATAALIGN_LEFT</i>	MSB alignment
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 data alignment */
```

```
adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);
```

### adc\_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(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t length);
<b>Function descriptions</b>	configure the length of regular channel group or inserted channel group
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
<b>adc_channel_group</b>	select the channel group
<i>ADC_REGULAR_CHANNEL</i>	regular channel group

<code>ADC_INSERTED_CHANNEL</code>	inserted channel group
<b>Input parameter{in}</b>	
<b>length</b>	the length of the channel, regular channel 1-16, inserted channel 1-4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the length of ADC0 regular channel */
```

```
adc_channel_length_config(ADC0, ADC_REGULAR_CHANNEL, 4);
```

### adc\_regular\_channel\_config

The description of `adc_regular_channel_config` is shown as below:

**Table 3-18. Function `adc_regular_channel_config`**

<b>Function name</b>	<code>adc_regular_channel_config</code>
<b>Function prototype</b>	<code>void adc_regular_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);</code>
<b>Function descriptions</b>	configure ADC regular channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<code>ADCx(x=0..2)</code>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>rank</b>	the regular group sequence rank, this parameter must be between 0 to 15
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the selected ADC channel
<code>ADC_CHANNEL_x(x=0..17)</code>	ADC Channelx (x=0..17)(x=16 and x=17 are only for ADC0)



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

Example:

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

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

### **adc\_inserted\_channel\_config**

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

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

<b>Function name</b>	adc_inserted_channel_config
<b>Function prototype</b>	void adc_inserted_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);

<b>Function descriptions</b>	configure ADC inserted channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(0,1)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>rank</b>	the inserted group sequencer rank, this parameter must be between 0 to 3
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the selected ADC channel
<i>ADC_CHANNEL_x(x=0..17)</i>	ADC Channelx (x=0..17)(x=16 and x=17 are only for ADC0)
<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_7 POINT5</i>	7.5 cycles
<i>ADC_SAMPLETIME_1 3POINT5</i>	13.5 cycles
<i>ADC_SAMPLETIME_2 8POINT5</i>	28.5 cycles
<i>ADC_SAMPLETIME_4 1POINT5</i>	41.5 cycles
<i>ADC_SAMPLETIME_5 5POINT5</i>	55.5 cycles
<i>ADC_SAMPLETIME_7 1POINT5</i>	71.5 cycles
<i>ADC_SAMPLETIME_2 39POINT5</i>	239.5 cycles
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

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

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

### adc\_inserted\_channel\_offset\_config

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

**Table 3-20. Function adc\_inserted\_channel\_offset\_config**

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

Example:

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

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

### adc\_external\_trigger\_config

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

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

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

Example:

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

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

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

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

**Table 3-22. 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(uint32_t adc_periph, uint8_t adc_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_periph</b>	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_channel_group</b>	select the channel group
ADC_REGULAR_CHANNEL	regular channel group
ADC_INSERTED_CHANNEL	inserted channel group
<b>Input parameter{in}</b>	
<b>external_trigger_source</b>	regular or inserted group trigger source
ADC0_1_EXTTRIGGER_REGULAR_T0_CH0	TIMER0 CH0 event select for regular channel
ADC0_1_EXTTRIGGER_REGULAR_T0_CH1	TIMER0 CH1 event select for regular channel
ADC0_1_EXTTRIGGER_REGULAR_T0_CH2	TIMER0 CH2 event select for regular channel
ADC0_1_EXTTRIGGER_REGULAR_T1_CH1	TIMER1 CH1 event select for regular channel

ADC0_1_EXTTRIG_REGULAR_T2_TRGO	TIMER2 TRGO event select for regular channel
ADC0_1_EXTTRIG_REGULAR_T3_CH3	TIMER3 CH3 event select for regular channel
ADC0_1_EXTTRIG_REGULAR_T7_TRGO	TIMER7 TRGO event select for regular channel
ADC0_1_EXTTRIG_REGULAR_EXTI_11	external interrupt line 11 for regular channel
ADC2_EXTTRIG_REGULAR_T2_CH0	TIMER2 CH0 event select for regular channel
ADC2_EXTTRIG_REGULAR_T1_CH2	TIMER1 CH2 event select for regular channel
ADC2_EXTTRIG_REGULAR_T0_CH2	TIMER0 CH2 event select for regular channel
ADC2_EXTTRIG_REGULAR_T7_CH0	TIMER7 CH0 event select for regular channel
ADC2_EXTTRIG_REGULAR_T7_TRGO	TIMER7 TRGO event select for regular channel
ADC2_EXTTRIG_REGULAR_T4_CH0	TIMER4 CH0 event select for regular channel
ADC2_EXTTRIG_REGULAR_T4_CH2	TIMER4 CH2 event select for regular channel
ADC0_1_2_EXTTRIG_REGULAR_NONE	software trigger for regular channel
ADC0_1_EXTTRIG_INSERTED_T0_TRGO	TIMER0 TRGO event select for inserted channel
ADC0_1_EXTTRIG_INSERTED_T0_CH3	TIMER0 CH3 event select for inserted channel
ADC0_1_EXTTRIG_INSERTED_T1_TRGO	TIMER1 TRGO event select for inserted channel
ADC0_1_EXTTRIG_INSERTED_T1_CH0	TIMER1 CH0 event select for inserted channel
ADC0_1_EXTTRIG_INSERTED_T2_CH3	TIMER2 CH3 event select for inserted channel

<i>ADC0_1_EXTTRIG_INSERTED_T3_TRGO</i>	TIMER3 TRGO event select for inserted channel
<i>ADC0_1_EXTTRIG_INSERTED_EXTI_15</i>	external interrupt line 15 for inserted channel
<i>ADC0_1_EXTTRIG_INSERTED_T7_CH3</i>	TIMER7 CH3 event select for inserted channel
<i>ADC2_EXTTRIG_INSERTED_T0_TRGO</i>	TIMER0 TRGO event select for inserted channel
<i>ADC2_EXTTRIG_INSERTED_T0_CH3</i>	TIMER0 CH3 event select for inserted channel
<i>ADC2_EXTTRIG_INSERTED_T3_CH2</i>	TIMER3 CH2 event select for inserted channel
<i>ADC2_EXTTRIG_INSERTED_T7_CH1</i>	TIMER7 CH1 event select for inserted channel
<i>ADC2_EXTTRIG_INSERTED_T7_CH3</i>	TIMER7 CH3 event select for injected channel
<i>ADC2_EXTTRIG_INSERTED_T4_TRGO</i>	TIMER4 TRGO event select for injected channel
<i>ADC2_EXTTRIG_INSERTED_T4_CH3</i>	TIMER4 CH3 event select for injected channel
<i>ADC0_1_2_EXTTRIG_INSERTED_NONE</i>	software trigger for injected channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC0 regular channel external trigger source */
adc_external_trigger_source_config(ADC0,ADC_REGULAR_CHANNEL,
ADC0_1_EXTTRIG_REGULAR_T0_CH0);
```

### **adc\_software\_trigger\_enable**

The description of `adc_software_trigger_enable` is shown as below:

Table 3-23. 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(uint32_t adc_periph, 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_periph</b>	ADC peripheral
<code>ADCx(x=0..2)</code>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_channel_group</b>	select the channel group
<code>ADC_REGULAR_CHANNEL</code>	regular channel group
<code>ADC_INSERTED_CHANNEL</code>	inserted channel group
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **adc\_regular\_data\_read**

The description of `adc_regular_data_read` is shown as below:

Table 3-24. Function `adc_regular_data_read`

<b>Function name</b>	<code>adc_regular_data_read</code>
<b>Function prototype</b>	<code>uint16_t adc_regular_data_read(uint32_t adc_periph);</code>
<b>Function descriptions</b>	read ADC regular group data register



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	ADC conversion value (0-0xFFFF)

Example:

```
/* read ADC0 regular group data register */
uint16_t adc_value = 0;
adc_value = adc_regular_data_read(ADC0);
```

### adc\_inserted\_data\_read

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

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

<b>Function name</b>	adc_inserted_data_read
<b>Function prototype</b>	uint16_t adc_inserted_data_read(uint32_t adc_periph, 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>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<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

Output parameter{out}	
-	-
Return value	
uint16_t	ADC conversion value (0-0xFFFF)

Example:

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

```
uint16_t adc_value = 0;
```

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

### adc\_sync\_mode\_convert\_value\_read

The description of adc\_sync\_mode\_convert\_value\_read is shown as below:

**Table 3-26. Function adc\_sync\_mode\_convert\_value\_read**

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

Example:

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

```
uint32_t adc_value = 0;
```

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

### adc\_flag\_get

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

Table 3-27. Function `adc_flag_get`

Function name	<code>adc_flag_get</code>
Function prototype	<code>FlagStatus adc_flag_get(uint32_t adc_periph , uint32_t adc_flag);</code>
Function descriptions	get the ADC flag bits
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC peripheral
<code>ADCx(x=0..2)</code>	ADC peripheral selection
Input parameter{in}	
<code>adc_flag</code>	the adc flag bits
<code>ADC_FLAG_WDE</code>	analog watchdog event flag
<code>ADC_FLAG_EOC</code>	end of group conversion flag
<code>ADC_FLAG_EOIC</code>	end of inserted group conversion flag
<code>ADC_FLAG_STIC</code>	start flag of inserted channel group
<code>ADC_FLAG_STRC</code>	start flag of regular channel group
Output parameter{out}	
-	-
Return value	
<code>FlagStatus</code>	SET or RESET

Example:

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

```
FlagStatus flag_value;
```

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

### `adc_flag_clear`

The description of `adc_flag_clear` is shown as below:

Table 3-28. Function `adc_flag_clear`

Function name	<code>adc_flag_clear</code>
---------------	-----------------------------

<b>Function prototype</b>	void adc_flag_clear(uint32_t adc_periph, uint32_t adc_flag);
<b>Function descriptions</b>	clear the ADC flag bits
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_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 regular channel group
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_flag_clear(ADC0, ADC_FLAG_WDE);
```

### adc\_interrupt\_flag\_get

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

**Table 3-29. Function adc\_interrupt\_flag\_get**

<b>Function name</b>	adc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, uint32_t adc_interrupt);
<b>Function descriptions</b>	get the ADC interrupt bits

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_interrupt</b>	the adc interrupt bits
<i>ADC_INT_FLAG_WDE</i>	analog watchdog interrupt
<i>ADC_INT_FLAG_EOC</i>	end of group conversion interrupt
<i>ADC_INT_FLAG_EOIC</i>	end of inserted group conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
FlagStatus flag_value;
```

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

### adc\_interrupt\_flag\_clear

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

**Table 3-30. Function adc\_interrupt\_flag\_clear**

<b>Function name</b>	adc_interrupt_flag_clear
<b>Function prototype</b>	void adc_interrupt_flag_clear(uint32_t adc_periph, uint32_t adc_interrupt);
<b>Function descriptions</b>	clear the ADC interrupt bits
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral

<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_interrupt</b>	the adc interrupt bits
<i>ADC_INT_FLAG_WDE</i>	analog watchdog interrupt
<i>ADC_INT_FLAG_EOC</i>	end of group conversion interrupt
<i>ADC_INT_FLAG_EOIC</i>	end of inserted group conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### adc\_interrupt\_enable

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

**Table 3-31. Function adc\_interrupt\_enable**

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

Example:

```
/* enable ADC0 analog watchdog interrupt */
```

```
adc_interrupt_enable(ADC0, ADC_INT_WDE);
```

### adc\_interrupt\_disable

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

**Table 3-32. Function adc\_interrupt\_disable**

<b>Function name</b>	adc_interrupt_disable
<b>Function prototype</b>	void adc_interrupt_disable(uint32_t adc_periph , uint32_t adc_interrupt);
<b>Function descriptions</b>	Disable ADC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC0 interrupt */
adc_interrupt_disable(ADC0, ADC_INT_WDE);
```

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

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

**Table 3-33. 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(uint32_t adc_periph, 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_periph</b>	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the selected ADC channel
ADC_CHANNEL_x(x=0..17)	ADC Channelx(x=0..17) (x=16 and x=17 are only for ADC0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### adc\_watchdog\_group\_channel\_enable

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



Table 3-34. Function `adc_watchdog_group_channel_enable`

<b>Function name</b>	<code>adc_watchdog_group_channel_enable</code>
<b>Function prototype</b>	<code>void adc_watchdog_group_channel_enable(uint32_t adc_periph, uint8_t adc_channel_group);</code>
<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_periph</b>	ADC peripheral
<code>ADCx(x=0..2)</code>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_channel_group</b>	the channel group use analog watchdog
<code>ADC_REGULAR_CHANNEL</code>	regular channel group
<code>ADC_INSERTED_CHANNEL</code>	inserted channel group
<code>ADC_REGULAR_INSERTED_CHANNEL</code>	both regular and inserted group
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **adc\_watchdog\_disable**

The description of `adc_watchdog_disable` is shown as below:

Table 3-35. Function `adc_watchdog_disable`

<b>Function name</b>	<code>adc_watchdog_disable</code>
<b>Function prototype</b>	<code>void adc_watchdog_disable(uint32_t adc_periph);</code>

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

Example:

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

```
adc_watchdog_disable(ADC0);
```

### adc\_watchdog\_threshold\_config

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

**Table 3-36. Function adc\_watchdog\_threshold\_config**

<b>Function name</b>	adc_watchdog_threshold_config
<b>Function prototype</b>	void adc_watchdog_threshold_config(uint32_t adc_periph, 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>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<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 ADC0 analog watchdog threshold */
```

```
adc_watchdog_threshold_config(ADC0, 0x0400, 0x0A00);
```

### adc\_oversample\_mode\_config

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

**Table 3-37. Function adc\_oversample\_mode\_config**

<b>Function name</b>	adc_oversample_mode_config
<b>Function prototype</b>	void adc_oversample_mode_config(uint32_t adc_periph, 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>adc_periph</b>	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
<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	no oversampling shift

<code>_SHIFT_NONE</code>	
<code>ADC_OVERSAMPLING_SHIFT_1B</code>	1-bit oversampling shift
<code>ADC_OVERSAMPLING_SHIFT_2B</code>	2-bit oversampling shift
<code>ADC_OVERSAMPLING_SHIFT_3B</code>	3-bit oversampling shift
<code>ADC_OVERSAMPLING_SHIFT_4B</code>	4-bit oversampling shift
<code>ADC_OVERSAMPLING_SHIFT_5B</code>	5-bit oversampling shift
<code>ADC_OVERSAMPLING_SHIFT_6B</code>	6-bit oversampling shift
<code>ADC_OVERSAMPLING_SHIFT_7B</code>	7-bit oversampling shift
<code>ADC_OVERSAMPLING_SHIFT_8B</code>	8-bit oversampling shift
<b>Input parameter{in}</b>	
<b>ratio</b>	ADC oversampling ratio
<code>ADC_OVERSAMPLING_RATIO_MUL2</code>	oversampling ratio multiple 2
<code>ADC_OVERSAMPLING_RATIO_MUL4</code>	oversampling ratio multiple 4
<code>ADC_OVERSAMPLING_RATIO_MUL8</code>	oversampling ratio multiple 8
<code>ADC_OVERSAMPLING_RATIO_MUL16</code>	oversampling ratio multiple 16
<code>ADC_OVERSAMPLING_RATIO_MUL32</code>	oversampling ratio multiple 32
<code>ADC_OVERSAMPLING_RATIO_MUL64</code>	oversampling ratio multiple 64
<code>ADC_OVERSAMPLING_RATIO_MUL128</code>	oversampling ratio multiple 128
<code>ADC_OVERSAMPLING_RATIO_MUL256</code>	oversampling ratio multiple 256

<code>_RATIO_MUL256</code>	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

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

adc_oversample_mode_config(ADC1, ADC_OVERSAMPLING_ALL_CONVERT,
ADC_OVERSAMPLING_SHIFT_4B, ADC_OVERSAMPLING_RATIO_MUL16);

```

### adc\_oversample\_mode\_enable

The description of `adc_oversample_mode_enable` is shown as below:

**Table 3-38. Function `adc_oversample_mode_enable`**

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

Example:

```

/* enable ADC0 oversample mode */

adc_oversample_mode_enable (ADC0);

```

## adc\_oversample\_mode\_disable

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

**Table 3-39. Function adc\_oversample\_mode\_disable**

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

Example:

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

## 3.3. BKP

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

### 3.3.1. Descriptions of Peripheral registers

BKP registers are listed in the table shown as below:

**Table 3-40. BKP Registers**

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

### 3.3.2. Descriptions of Peripheral functions

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

**Table 3-41. BKP firmware function**

Function name	Function description
bkp_deinit	reset data registers
bkp_write_data	write BKP data register
bkp_read_data	read BKP data register
bkp_rtc_calibration_output_enable	enable RTC clock calibration output
bkp_rtc_calibration_output_disable	disable RTC clock calibration output
bkp_rtc_signal_output_enable	enable RTC alarm or second signal output
bkp_rtc_signal_output_disable	disable RTC alarm or second signal output
bkp_rtc_output_select	select RTC output, the RTC output can be select as alarm pulse or second pulse
bkp_rtc_clock_output_select	select RTC clock output
bkp_rtc_clock_calibration_direction	select RTC clock calibration direction
bkp_rtc_calibration_value_set	set RTC clock calibration value
bkp_tamper_detection_enable	enable tamper detection
bkp_tamper_detection_disable	disable tamper detection
bkp_tamper_active_level_set	set tamper pin active level
bkp_tamper_interrupt_enable	enable tamper interrupt
bkp_tamper_interrupt_disable	disable tamper interrupt
bkp_flag_get	get bkp flag state

Function name	Function description
bkp_flag_clear	clear bkp flag state
bkp_interrupt_flag_get	get bkp interrupt flag state
bkp_interrupt_flag_clear	clear bkp interrupt flag state

## bkp\_deinit

The description of bkp\_deinit is shown as below:

**Table 3-42. Function bkp\_deinit**

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

Example:

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

```
bkp_deinit ();
```

## bkp\_write\_data

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

**Table 3-43. Function bkp\_write\_data**

Function name	bkp_write_data
Function prototype	void bkp_write_data(bkp_data_register_enum register_number, uint16_t data);
Function descriptions	write BKP data register



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>register_number</b>	refer to bkp_data_register_enum
<i>BKP_DATA_x</i> ( <i>x</i> = 0..41)	bkp data register number <i>x</i>
<b>Input parameter{in}</b>	
<b>data</b>	the data to be write in BKP data register
<i>0-0xffff</i>	data value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### bkp\_read\_data

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

**Table 3-44. Function bkp\_data\_read**

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

-	-
Return value	
uint16_t	0-0xffff

Example:

```
/* read BKP data register */
```

```
uint16_t data;
```

```
data = bkp_read_data (BKP_DATA_0);
```

### bkp\_rtc\_calibration\_output\_enable

The description of bkp\_rtc\_calibration\_output\_enable is shown as below:

**Table 3-45. Function bkp\_rtc\_calibration\_output\_enable**

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

Example:

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

```
bkp_rtc_calibration_output_enable();
```

### bkp\_rtc\_calibration\_output\_disable

The description of bkp\_rtc\_calibration\_output\_disable is shown as below:

**Table 3-46. Function bkp\_rtc\_calibration\_output\_disable**

Function name	bkp_rtc_calibration_output_disable
---------------	------------------------------------

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

Example:

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

```
bkp_rtc_calibration_output_disable();
```

### bkp\_rtc\_signal\_output\_enable

The description of bkp\_rtc\_signal\_output\_enable is shown as below:

**Table 3-47. Function bkp\_rtc\_signal\_output\_enable**

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

Example:

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

```
bkp_rtc_signal_output_enable();
```

### bkp\_rtc\_signal\_output\_disable

The description of bkp\_rtc\_signal\_output\_disable is shown as below:

**Table 3-48. Function bkp\_rtc\_signal\_output\_disable**

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

Example:

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

```
bkp_rtc_signal_output_disable();
```

### bkp\_rtc\_output\_select

The description of bkp\_rtc\_output\_select is shown as below:

**Table 3-49. Function bkp\_rtc\_output\_select**

<b>Function name</b>	bkp_rtc_output_select
<b>Function prototype</b>	void bkp_rtc_output_select (uint16_t outputsel);
<b>Function descriptions</b>	select RTC output, the RTC output can be select as alarm pulse or second pulse
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>outputsel</b>	RTC output selection
<i>RTC_OUTPUT_ALARM_PULSE</i>	RTC alarm pulse is selected as the RTC output
<i>RTC_OUTPUT_SECONDS_PULSE</i>	RTC second pulse is selected as the RTC output
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select RTC output alarm signal output */
```

```
bkp_rtc_output_select (RTC_OUTPUT_ALARM_PULSE);
```

### bkp\_rtc\_clock\_output\_select

The description of bkp\_rtc\_clock\_output\_select is shown as below:

**Table 3-50. Function bkp\_rtc\_clock\_output\_select**

<b>Function name</b>	bkp_rtc_clock_output_select
<b>Function prototype</b>	void bkp_rtc_clock_output_select(uint16_t clocksel);
<b>Function descriptions</b>	select RTC clock output, the RTC clock output can be select as divided 64 or no division
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>clocksel</b>	RTC clock output selection
<i>RTC_CLOCK_DIV_64</i>	RTC clock divided 64 is selected as the RTC clock output
<i>RTC_CLOCK_DIV_1</i>	RTC clock is selected as the RTC clock output
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* select RTC clock divided 64 to output */
```

```
bkp_rtc_clock_output_select (RTC_CLOCK_DIV_64);
```

### bkp\_rtc\_clock\_calibration\_direction

The description of bkp\_rtc\_clock\_calibration\_direction is shown as below:

**Table 3-51. Function bkp\_rtc\_clock\_calibration\_direction**

<b>Function name</b>	bkp_rtc_clock_calibration_direction
<b>Function prototype</b>	void bkp_rtc_clock_calibration_direction(uint16_t direction);
<b>Function descriptions</b>	select RTC clock calibration direction, the RTC clock calibration direction can be select as slowed down or speed up
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>direction</b>	RTC clock calibration direction
RTC_CLOCK_SLOWED_DOWN	RTC clock slowed down
RTC_CLOCK_SPEED_UP	RTC clock speed up
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
bkp_rtc_clock_calibration_direction (RTC_CLOCK_SLOWED_DOWN);
```

### bkp\_rtc\_calibration\_value\_set

The description of bkp\_rtc\_calibration\_value\_set is shown as below:

Table 3-52. Function bkp\_rtc\_calibration\_value\_set

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

Example:

```
/* set RTC clock calibration value */
bkp_rtc_calibration_value_set (0x7f);
```

### bkp\_tamper\_detection\_enable

The description of bkp\_tamper\_detection\_enable is shown as below:

Table 3-53. Function bkp\_tamper\_detection\_enable

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

Return value	
-	-

Example:

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

### bkp\_tamper\_detection\_disable

The description of bkp\_tamper\_detection\_disable is shown as below:

**Table 3-54. Function bkp\_tamper\_detection\_disable**

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

Example:

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

### bkp\_tamper\_active\_level\_set

The description of bkp\_tamper\_active\_level\_set is shown as below:

**Table 3-55. Function bkp\_tamper\_active\_level\_set**

Function name	bkp_tamper_active_level_set
Function prototype	void bkp_tamper_active_level_set (uint16_t level);
Function descriptions	set tamper pin active level



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>level</b>	tamper pin active level
<i>TAMPER_PIN_ACTIVE_HIGH</i>	the tamper pin is active high
<i>TAMPER_PIN_ACTIVE_LOW</i>	the tamper pin is active low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
bkp_tamper_active_level_set (TAMPER_PIN_ACTIVE_HIGH);
```

### bkp\_tamper\_interrupt\_enable

The description of bkp\_tamper\_interrupt\_enable is shown as below:

**Table 3-56. Function bkp\_tamper\_interrupt\_enable**

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

Example:

```
/* enable tamper pin interrupt */
```

```
bkp_tamper_interrupt_enable ();
```

### bkp\_tamper\_interrupt\_disable

The description of bkp\_tamper\_interrupt\_disable is shown as below:

**Table 3-57. Function bkp\_tamper\_interrupt\_disable**

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

Example:

```
/* disable tamper pin interrupt */
```

```
bkp_tamper_interrupt_disable ();
```

### bkp\_flag\_get

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

**Table 3-58. Function bkp\_flag\_get**

Function name	bkp_flag_get
Function prototype	FlagStatus bkp_flag_get(uint16_t flag);
Function descriptions	get bkp flag state
Precondition	-
The called functions	-

Input parameter{in}	
<b>flag</b>	bkp flag state
<i>BKP_FLAG_TAMPE</i> <i>R</i>	tamper event flag
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
FlagStatus status;
```

```
status = bkp_flag_get (BKP_FLAG_TAMPER);
```

### bkp\_flag\_clear

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

**Table 3-59. Function bkp\_flag\_clear**

<b>Function name</b>	bkp_flag_clear
<b>Function prototype</b>	void bkp_flag_clear(uint16_t flag);
<b>Function descriptions</b>	clear bkp flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>flag</b>	bkp flag state
<i>BKP_FLAG_TAMPE</i> <i>R</i>	tamper event flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear BKP flag state */
```

```
bkp_flag_clear (BKP_FLAG_TAMPER);
```

### bkp\_interrupt\_flag\_get

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

**Table 3-60. Function bkp\_interrupt\_flag\_get**

<b>Function name</b>	bkp_interrupt_flag_get
<b>Function prototype</b>	FlagStatus bkp_interrupt_flag_get(uint16_t flag);
<b>Function descriptions</b>	get bkp interrupt flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	bkp interrupt flag state
<i>BKP_INT_FLAG_TAMPER</i>	tamper interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
bkp_interrupt_flag_get (BKP_INT_FLAG_TAMPER);
```

### bkp\_interrupt\_flag\_clear

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

**Table 3-61. Function bkp\_interrupt\_flag\_clear**

<b>Function name</b>	bkp_interrupt_flag_clear
<b>Function prototype</b>	void bkp_interrupt_flag_clear(uint16_t flag);
<b>Function descriptions</b>	clear bkp interrupt flag state
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	bkp interrupt flag state
<i>BKP_INT_FLAG_TAMPER</i>	tamper interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear BKP interrupt flag state */
```

```
bkp_interrupt_flag_clear (BKP_INT_FLAG_TAMPER);
```

## 3.4. CAN

CAN bus (for Controller Area Network) is a bus standard designed to allow microcontrollers and devices to communicate with each other without a host computer. The CAN registers are listed in chapter [3.4.1](#), the CAN firmware functions are introduced in chapter [3.4.2](#).

### 3.4.1. Descriptions of Peripheral registers

CAN registers are listed in the table shown as below:

**Table 3-62. CAN Registers**

Registers	Descriptions
CAN_CTL	Control register
CAN_STAT	Status register
CAN_TSTAT	Transmit status register
CAN_RFIFO0	Receive message FIFO0 register
CAN_RFIFO1	Receive message FIFO1 register
CAN_INTEN	Interrupt enable register
CAN_ERR	Error register
CAN_BT	Bit timing register

Registers	Descriptions
CAN_TMIx	Transmit mailbox identifier register
CAN_TMPx	Transmit mailbox property register
CAN_TMDATA0x	Transmit mailbox data0 register
CAN_TMDATA1x	Transmit mailbox data1 register
CAN_RFIFOMIx	Receive FIFO mailbox identifier register
CAN_RFIFOMPx	Receive FIFO mailbox property register
CAN_RFIFOMDAT A0x	Receive FIFO mailbox data0 register
CAN_RFIFOMDAT A1x	Receive FIFO mailbox data1 register
CAN_FCTL	Filter control register
CAN_FMCFG	Filter mode configuration register
CAN_FSCFG	Filter scale configuration register
CAN_FAFIFO	Filter associated FIFO registe
CAN_FW	Filter working register
CAN_FxDATAy	Filter x data y register

### 3.4.2. Descriptions of Peripheral functions

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

**Table 3-63. CAN firmware function**

Function name	Function description
can_deinit	deinitialize CAN
can_struct_para_init	initialize CAN parameter struct with a default value
can_init	initialize CAN
can_filter_init	initialize CAN filter
can1_filter_start_bank	set can1 fliter start bank number
can_debug_freeze_enable	CAN debug freeze enable
can_debug_freeze_disable	CAN debug freeze disable

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

## Structure can\_parameter\_struct

Table 3-64. Structure can\_parameter\_struct

Member name	Function description
working_mode	CAN working mode
resync_jump_width	CAN resynchronization jump width
time_segment_1	time segment 1
time_segment_2	time segment 2
time_triggered	time triggered communication mode

auto_bus_off_recovery	automatic bus-off recovery
auto_wake_up	automatic wake-up mode
auto_retrans	automatic retransmission mode
rec_fifo_overwrite	receive FIFO overwrite mode
trans_fifo_order	transmit FIFO order
prescaler	baudrate prescaler

### Structure can\_trasnmit\_message\_struct

**Table 3-65. Structure can\_trasnmit\_message\_struct**

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

### Structure can\_receive\_message\_struct

**Table 3-66. Structure can\_receive\_message\_struct**

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



## Structure can\_filter\_parameter\_struct

Table 3-67. Structure can\_filter\_parameter\_struct

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

## can\_deinit

The description of can\_deinit is shown as below:

Table 3-68. Function can\_deinit

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

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

Example:

```
/* CAN0 deinitialize*/
```

```
can_deinit (CAN0);
```

### can\_struct\_para\_init

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

**Table 3-69. Function can\_struct\_para\_init**

<b>Function name</b>	can_struct_para_init
<b>Function prototype</b>	void can_struct_para_init(can_struct_type_enum type, void* p_struct)
<b>Function descriptions</b>	initialize CAN parameter struct with a default value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>type</b>	CAN peripheral
<i>CAN_INIT_STRUCT</i>	CAN initialize parameters struct
<i>CAN_FILTER_STRUCT</i>	CAN filter parameters struct
<i>CAN_TX_MESSAGE_STRUCT</i>	CAN transmit message struct

<code>CAN_RX_MESSAGE_STRUCT</code>	CAN receive message struct
<b>Output parameter{out}</b>	
<b>p_struct</b>	the struct pointer that needs initialize
<b>Return value</b>	
-	-

Example:

```
can_parameter_struct can_init;
```

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

### can\_init

The description of can\_init is shown as below:

**Table 3-70. Function can\_init**

<b>Function name</b>	can_init
<b>Function prototype</b>	ErrStatus can_init(uint32_t can_periph, can_parameter_struct* can_parameter_init);
<b>Function descriptions</b>	initialize CAN
<b>Precondition</b>	can_struct_para_init()
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<code>CANx(x=0,1)</code>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>can_parameter_init</b>	CAN parameter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-64. Structure can_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS / ERROR

<b>Function name</b>	can_init
<b>Function prototype</b>	ErrStatus can_init(uint32_t can_periph, can_parameter_struct* can_parameter_init);
<b>Function descriptions</b>	initialize CAN
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
<b>Input parameter{in}</b>	
can_parameter_init	CAN parameter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-64. Structure can_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
ErrStatus	SUCCESS / ERROR

Example:

```
/* CAN0 initialize*/
```

```
can_init (CAN0);
```

### can\_filter\_init

The description of can\_filter\_init is shown as below:

**Table 3-71. Function can\_filter\_init**

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

<b>can_filter_parameter_init</b>	CAN filter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-67. Structure can filter parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-
<b>Function name</b>	can_filter_init
<b>Function prototype</b>	void can_filter_init(can_filter_parameter_struct* can_filter_parameter_init);
<b>Function descriptions</b>	initialize CAN filter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_filter_parameter_init</b>	CAN filter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-67. Structure can filter parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### can1\_filter\_start\_bank

The description of can1\_filter\_start\_bank is shown as below:

**Table 3-72. Function can1\_filter\_start\_bank**

<b>Function name</b>	can1_filter_start_bank
<b>Function prototype</b>	void can1_filter_start_bank(uint8_t start_bank);
<b>Function descriptions</b>	set CAN1 filter start bank number
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>start_bank</b>	CAN1 start bank number
1..27	start number
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
can1_filter_start_bank (15);
```

### can\_debug\_freeze\_enable

The description of can\_debug\_freeze\_enable is shown as below:

**Table 3-73. Function can\_debug\_freeze\_enable**

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

<b>Precondition</b>	-
<b>The called functions</b>	dbg_periph_enable
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable CAN0 debug freeze */
```

```
can_debug_freeze_enable (CAN0);
```

### can\_debug\_freeze\_disable

The description of can\_debug\_freeze\_disable is shown as below:

**Table 3-74. Function can\_debug\_freeze\_disable**

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

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

Example:

```
/* disable CAN0 debug freeze */
```

```
can_debug_freeze_disable (CAN0);
```

### can\_time\_trigger\_mode\_enable

The description of can\_time\_trigger\_mode\_enable is shown as below:

**Table 3-75. Function can\_time\_trigger\_mode\_enable**

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



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

Example:

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

### can\_time\_trigger\_mode\_disable

The description of can\_time\_trigger\_mode\_disable is shown as below:

**Table 3-76. Function can\_time\_trigger\_mode\_disable**

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

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

Example:

```
/* disable CAN0 time trigger mode */
```

```
can_time_trigger_mode_disable (CAN0);
```

### can\_message\_transmit

The description of can\_message\_transmit is shown as below:

**Table 3-77. Function can\_message\_transmit**

<b>Function name</b>	can_message_transmit
<b>Function prototype</b>	uint8_t can_message_transmit(uint32_t can_periph, can_transmit_message_struct* transmit_message);
<b>Function descriptions</b>	transmit CAN message
<b>Precondition</b>	can_struct_para_init()
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>transmit_message</b>	CAN transmit message struct, the structure members can refer to members of the structure <a href="#">Table 3-65. Structure <i>can_transmit_message</i> struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	0x00-0x03
<b>Function name</b>	can_message_transmit
<b>Function prototype</b>	uint8_t can_message_transmit(uint32_t can_periph, can_transmit_message_struct* transmit_message);
<b>Function descriptions</b>	transmit CAN message
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>transmit_message</b>	CAN transmit message struct, the structure members can refer to members of the structure <a href="#">Table 3-65. Structure <i>can_transmit_message</i> struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	0x00-0x03

Example:

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

```
uint8_t transmit_mailbox = 0;
```

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

## can\_transmit\_states

The description of can\_transmit\_states is shown as below:

**Table 3-78. Function can\_transmit\_states**

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

<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>mailbox_number</b>	Mailbox number
<i>CAN_MAILBOXx</i>	<i>CAN_MAILBOXx(x=0,1,2)</i>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>can_transmit_state_e num</b>	0..4

Example:

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

```
uint8_t transmit_state = 0;
```

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

### can\_transmission\_stop

The description of can\_transmission\_stop is shown as below:

**Table 3-79. Function can\_transmission\_stop**

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

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

Example:

```
/* stop CAN0 mailbox0 transmission */
can_transmission_stop (CAN0, CAN_MAILBOX0);
```

### can\_message\_receive

The description of can\_message\_receive is shown as below:

**Table 3-80. Function can\_message\_receive**

<b>Function name</b>	can_message_receive
<b>Function prototype</b>	void can_message_receive(uint32_t can_periph, uint8_t fifo_number, can_receive_message_struct* receive_message);
<b>Function descriptions</b>	CAN receive message

<b>Precondition</b>	can_struct_para_init()
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>fifo_number</b>	Fifo number
<i>CAN_FIFOx</i>	CAN_FIFOx(x=0,1)
<b>Input parameter{in}</b>	
<b>receive_message</b>	CAN message receive struct, the structure members can refer to members of the structure <a href="#">Table 3-66. Structure can_receive_message_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-
<b>Function name</b>	can_message_receive
<b>Function prototype</b>	void can_message_receive(uint32_t can_periph, uint8_t fifo_number, can_receive_message_struct* receive_message);
<b>Function descriptions</b>	CAN receive message
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>fifo_number</b>	Fifo number
<i>CAN_FIFOx</i>	CAN_FIFOx(x=0,1)
<b>Input parameter{in}</b>	
<b>receive_message</b>	CAN message receive struct, the structure members can refer to members

	of the structure <a href="#">Table 3-66. Structure can_receive_message_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### can\_fifo\_release

The description of can\_fifo\_release is shown as below:

**Table 3-81. Function can\_fifo\_release**

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



<b>Function descriptions</b>	release FIFO0
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>fifo_number</b>	Fifo number
<i>CAN_FIFOx</i>	CAN_FIFOx(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### can\_receive\_message\_length\_get

The description of can\_receive\_message\_length\_get is shown as below:

**Table 3-82. Function can\_receive\_message\_length\_get**

<b>Function name</b>	can_receive_message_length_get
<b>Function prototype</b>	uint8_t can_receive_message_length_get(uint32_t can_periph, uint8_t fifo_number);
<b>Function descriptions</b>	CAN receive message length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection

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

Example:

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

```
uint8_t frame_number = 0;
```

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

## can\_working\_mode\_set

The description of can\_working\_mode\_set is shown as below:

**Table 3-83. Function can\_working\_mode\_set**

<b>Function name</b>	can_working_mode_set
<b>Function prototype</b>	ErrStatus can_working_mode_set(uint32_t can_periph, uint8_t working_mode);
<b>Function descriptions</b>	set CAN working mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>can_working_mode</b>	Mode select
CAN_MODE_INITIALIZE	Initialize mode
CAN_MODE_NORMAL	Normal mode
CAN_MODE_SLEEP	Sleep mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS / ERROR

Example:

```
/* set CAN0 working at initialize mode */
```

```
can_working_mode_set (CAN0, CAN_MODE_INITIALIZE);
```

## can\_wakeup

The description of can\_wakeup is shown as below:

**Table 3-84. Function can\_wakeup**

<b>Function name</b>	can_wakeup
----------------------	------------

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

Example:

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

### can\_error\_get

The description of can\_error\_get is shown as below:

Table 3-85. Function can\_error\_get

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

Example:

```
/* get CAN0 error type */
```

```
can_error_get (CAN0);
```

## can\_receive\_error\_number\_get

The description of can\_receive\_error\_number\_get is shown as below:

**Table 3-86. Function can\_receive\_error\_number\_get**

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

Example:

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

```
can_receive_error_number_get (CAN0);
```

### can\_transmit\_error\_number\_get it

The description of can\_transmit\_error\_number\_get is shown as below:

**Table 3-87. Function can\_transmit\_error\_number\_get**

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

uint8_t	0..255
---------	--------

Example:

```
/* get CAN0 transmit error number */
```

```
can_transmit_error_number_get (CAN0);
```

### can\_interrupt\_enable

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

**Table 3-88. Function can\_interrupt\_enable**

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



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

<i>CAN_INT_ERR</i>	error interrupt enable
<i>CAN_INT_WU</i>	wakeup interrupt enable
<i>CAN_INT_SLPW</i>	sleep working interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* CAN0 transmit mailbox empty interrupt enable */
```

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

### can\_interrupt\_disable

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

**Table 3-89. Function can\_interrupt\_disable**

<b>Function name</b>	can_interrupt_disable
<b>Function prototype</b>	void can_interrupt_disable(uint32_t can_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable CAN interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	Interrupt type
<i>CAN_INT_TME</i>	transmit mailbox empty interrupt enable
<i>CAN_INT_RFNE0</i>	receive FIFO0 not empty interrupt enable
<i>CAN_INT_RFF0</i>	receive FIFO0 full interrupt enable
<i>CAN_INT_RFO0</i>	receive FIFO0 overfull interrupt enable
<i>CAN_INT_RFNE1</i>	receive FIFO1 not empty interrupt enable

<i>CAN_INT_RFF1</i>	receive FIFO1 full interrupt enable
<i>CAN_INT_RFO1</i>	receive FIFO1 overfull interrupt enable
<i>CAN_INT_WERR</i>	warning error interrupt enable
<i>CAN_INT_PERR</i>	passive error interrupt enable
<i>CAN_INT_BO</i>	bus-off interrupt enable
<i>CAN_INT_ERRN</i>	error number interrupt enable
<i>CAN_INT_ERR</i>	error interrupt enable
<i>CAN_INT_WU</i>	wakeup interrupt enable
<i>CAN_INT_SLPW</i>	sleep working interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* CAN0 transmit mailbox empty interrupt disable */
```

```
can_interrupt_disable (CAN0, CAN_INT_TME);
```

### can\_flag\_get

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

**Table 3-90. Function can\_flag\_get**

<b>Function name</b>	can_flag_get
<b>Function prototype</b>	FlagStatus can_flag_get(uint32_t can_periph, can_flag_enum flag);
<b>Function descriptions</b>	get CAN flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	

flag	CAN flags
CAN_FLAG_MTE2	mailbox 2 transmit error
CAN_FLAG_MTE1	mailbox 1 transmit error
CAN_FLAG_MTE0	mailbox 0 transmit error
CAN_FLAG_MTF2	mailbox 2 transmit finished
CAN_FLAG_MTF1	mailbox 1 transmit finished
CAN_FLAG_MTF0	mailbox 0 transmit finished
CAN_FLAG_RFO0	receive FIFO0 overfull
CAN_FLAG_RFF0	receive FIFO0 full
CAN_FLAG_RFO1	receive FIFO1 overfull
CAN_FLAG_RFF1	receive FIFO1 full
CAN_FLAG_BOERR	bus-off error
CAN_FLAG_PERR	passive error
CAN_FLAG_WERR	warning error
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

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

```
can_flag_get (CAN0, CAN_FLAG_MTF0);
```

### can\_flag\_clear

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

**Table 3-91. Function can\_flag\_clear**

Function name	can_flag_clear
Function prototype	void can_flag_clear(uint32_t can_periph, can_flag_enum flag);
Function descriptions	clear CAN flag state
Precondition	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>flag</b>	CAN flags
<i>CAN_FLAG_MTE2</i>	mailbox 2 transmit error
<i>CAN_FLAG_MTE1</i>	mailbox 1 transmit error
<i>CAN_FLAG_MTE0</i>	mailbox 0 transmit error
<i>CAN_FLAG_MTF2</i>	mailbox 2 transmit finished
<i>CAN_FLAG_MTF1</i>	mailbox 1 transmit finished
<i>CAN_FLAG_MTF0</i>	mailbox 0 transmit finished
<i>CAN_FLAG_RFO0</i>	receive FIFO0 overfull
<i>CAN_FLAG_RFF0</i>	receive FIFO0 full
<i>CAN_FLAG_RFO1</i>	receive FIFO1 overfull
<i>CAN_FLAG_RFF1</i>	receive FIFO1 full
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear CAN0 mailbox 0 transmit error flag*/
```

```
can_flag_clear (CAN0, CAN_FLAG_MTE0);
```

### can\_interrupt\_flag\_get

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

**Table 3-92. Function can\_interrupt\_flag\_get**

<b>Function name</b>	can_interrupt_flag_get
<b>Function prototype</b>	FlagStatus can_interrupt_flag_get(uint32_t can_periph,

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

Example:

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

## can\_interrupt\_flag\_clear

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

**Table 3-93. Function can\_interrupt\_flag\_clear**

<b>Function name</b>	can_interrupt_flag_clear
<b>Function prototype</b>	void can_interrupt_flag_clear(uint32_t can_periph, can_interrupt_flag_enum flag);
<b>Function descriptions</b>	clear CAN interrupt flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>flag</b>	CAN interrupt flags
CAN_INT_FLAG_SLPIF	status change interrupt flag of sleep working mode entering
CAN_INT_FLAG_WUIF	status change interrupt flag of wakeup from sleep working mode
CAN_INT_FLAG_ERRIF	error interrupt flag
CAN_INT_FLAG_MTF2	mailbox 2 transmit finished interrupt flag
CAN_INT_FLAG_MTF1	mailbox 1 transmit finished interrupt flag
CAN_INT_FLAG_MTF0	mailbox 0 transmit finished interrupt flag
CAN_INT_FLAG_RFO0	receive FIFO0 overfull interrupt flag
CAN_INT_FLAG_RFF0	receive FIFO0 full interrupt flag
CAN_INT_FLAG_RFO1	receive FIFO1 overfull interrupt flag
CAN_INT_FLAG_RFF1	receive FIFO1 full interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

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

## 3.5. 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 [3.5.1](#), the CRC firmware functions are introduced in chapter [3.5.2](#).

### 3.5.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

**Table 3-94. CRC Registers**

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

### 3.5.2. Descriptions of Peripheral functions

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

**Table 3-95. CRC firmware function**

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



## crc\_deinit

The description of crc\_deinit is shown as below:

**Table 3-96. Function crc\_deinit**

<b>Function name</b>	crc_deinit
<b>Function prototype</b>	void crc_deinit(void);
<b>Function descriptions</b>	deinit CRC calculation unit
<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 */
crc_deinit();
```

## crc\_data\_register\_reset

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

**Table 3-97. 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>	

-	-
Return value	
-	-

Example:

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

### **crc\_data\_register\_read**

The description of `crc_data_register_read` is shown as below:

**Table 3-98. Function `crc_data_register_read`**

<b>Function name</b>	<code>crc_data_register_read</code>
<b>Function prototype</b>	<code>uint32_t crc_data_register_read(void);</code>
<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 3-99. Function `crc_free_data_register_read`**

<b>Function name</b>	<code>crc_free_data_register_read</code>
----------------------	--

<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 3-100. 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	specified 8-bit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* write the free data register */
crc_free_data_register_write(0x11);
```

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

The description of `crc_single_data_calculate` is shown as below:

**Table 3-101. Function `crc_single_data_calculate`**

<b>Function name</b>	<code>crc_single_data_calculate</code>
<b>Function prototype</b>	<code>uint32_t crc_single_data_calculate(uint32_t sdata);</code>
<b>Function descriptions</b>	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>	specified 32-bit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	32-bit value calculated by CRC (0-0xFFFFFFFF)

Example:

```
/* CRC calculate a 32-bit data */
uint32_t val = 0, valcrc = 0;
val = (uint32_t)0xabcd1234;
rcu_periph_clock_enable(RCU_CRC);
valcrc = crc_single_data_calculate(val);
```

### **crc\_block\_data\_calculate**

The description of `crc_block_data_calculate` is shown as below:

**Table 3-102. Function `crc_block_data_calculate`**

<b>Function name</b>	<code>crc_block_data_calculate</code>
<b>Function prototype</b>	<code>uint32_t crc_block_data_calculate(uint32_t array[], uint32_t size);</code>

<b>Function descriptions</b>	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 bit 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 value calculated by CRC (0-0xFFFFFFFF)

Example:

```

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

#define BUFFER_SIZE    6

uint32_t valcrc = 0;

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

rcu_periph_clock_enable(RCU_CRC);

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

```

## 3.6. CTC

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

### 3.6.1. Descriptions of Peripheral registers

CTC registers are listed in the table shown as below:

**Table 3-103. CTC Registers**

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

### 3.6.2. Descriptions of Peripheral functions

CTC registers are listed in the table shown as below:

**Table 3-104. CTC firmware function**

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

Function name	Function description
ctc_interrupt_disable	disable the CTC interrupt
ctc_interrupt_flag_get	get CTC interrupt flag
ctc_interrupt_flag_clear	clear CTC interrupt flag
ctc_flag_get	get CTC flag
ctc_flag_clear	clear CTC flag

### ctc\_deinit

The description of ctc\_deinit is shown as below:

**Table 3-105. Function ctc\_deinit**

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

Example:

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

### ctc\_counter\_enable

The description of ctc\_counter\_enable is shown as below:

**Table 3-106. Function ctc\_counter\_enable**

Function name	ctc_counter_enable
---------------	--------------------

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

Example:

```
/* enable CTC trim counter*/
```

```
ctc_counter_enable ();
```

### ctc\_counter\_disable

The description of ctc\_counter\_disable is shown as below:

**Table 3-107. Function ctc\_counter\_disable**

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

Example:



```
/* disable CTC trim counter */
```

```
ctc_counter_disable ();
```

### ctc\_irc48m\_trim\_value\_config

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

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

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

Example:

```
/* IRC48M trim value configuration */
```

```
ctc_irc48m_trim_value_config (0x01);
```

### ctc\_software\_refsource\_pulse\_generate

The description of ctc\_software\_refsource\_pulse\_generate is shown as below:

**Table 3-109. Function ctc\_software\_refsource\_pulse\_generate**

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

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

Example:

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

```
ctc_software_refsource_pulse_generate ();
```

### ctc\_hardware\_trim\_mode\_config

The description of ctc\_hardware\_trim\_mode\_config is shown as below:

**Table 3-110. Function ctc\_hardware\_trim\_mode\_config**

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

Example:

```
/* enable CTC hardware trim */
```

```
ctc_hardware_trim_mode_config (CTC_HARDWARE_TRIM_MODE_ENABLE);
```

## ctc\_refsource\_polarity\_config

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

**Table 3-111. Function ctc\_refsource\_polarity\_config**

<b>Function name</b>	ctc_refsource_polarity_config
<b>Function prototype</b>	void ctc_refsource_polarity_config(uint32_t polarity);
<b>Function descriptions</b>	configure reference signal source polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>polarity</b>	reference signal source polarity
<i>CTC_REFSOURCE_POLARITY_FALLING</i>	reference signal source polarity is falling edge
<i>CTC_REFSOURCE_POLARITY_RISING</i>	reference signal source polarity is rising edge
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
ctc_refsource_polarity_config (CTC_REFSOURCE_POLARITY_RISING);
```

## ctc\_refsource\_signal\_select

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

**Table 3-112. Function ctc\_refsource\_signal\_select**

<b>Function name</b>	ctc_refsource_signal_select
<b>Function prototype</b>	void ctc_refsource_signal_select(uint32_t refs);
<b>Function descriptions</b>	select reference signal source
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>refs</b>	reference signal source
<i>CTC_REFSOURCE_G PIO</i>	GPIO is selected
<i>CTC_REFSOURCE_LX TAL</i>	LXTAL is selected
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
ctc_refsource_signal_select (CTC_REFSOURCE_LXTAL);
```

### ctc\_refsource\_prescaler\_config

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

**Table 3-113. Function ctc\_refsource\_prescaler\_config**

<b>Function name</b>	ctc_refsource_prescaler_config
<b>Function prototype</b>	void ctc_refsource_prescaler_config(uint32_t prescaler);
<b>Function descriptions</b>	configure reference signal source prescaler
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>prescaler</b>	Prescaler factor
<i>CTC_REFSOURCE_P SC_OFF</i>	reference signal not divided
<i>CTC_REFSOURCE_P SC_DIV2</i>	reference signal divided by 2
<i>CTC_REFSOURCE_P SC_DIV4</i>	reference signal divided by 4
<i>CTC_REFSOURCE_P</i>	reference signal divided by 8

<i>SC_DIV8</i>	
<i>CTC_REFSOURCE_P</i> <i>SC_DIV16</i>	reference signal divided by 16
<i>CTC_REFSOURCE_P</i> <i>SC_DIV32</i>	reference signal divided by 32
<i>CTC_REFSOURCE_P</i> <i>SC_DIV64</i>	reference signal divided by 64
<i>CTC_REFSOURCE_P</i> <i>SC_DIV128</i>	reference signal divided by 128
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
ctc_refsource_prescaler_config(CTC_REFSOURCE_PSC_DIV2);
```

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

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

**Table 3-114. Function ctc\_clock\_limit\_value\_config**

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

Example:

```
/* configure clock trim base limit value */
```

```
ctc_clock_limit_value_config (0x1F);
```

### ctc\_counter\_reload\_value\_config

The description of ctc\_counter\_reload\_value\_config is shown as below:

**Table 3-115. Function ctc\_counter\_reload\_value\_config**

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

Example:

```
/* configure CTC counter reload value */
```

```
ctc_counter_reload_value_config (0x00FF);
```

### ctc\_counter\_capture\_value\_read

The description of ctc\_counter\_capture\_value\_read is shown as below:

**Table 3-116. Function ctc\_counter\_capture\_value\_read**

<b>Function name</b>	ctc_counter_capture_value_read
<b>Function prototype</b>	uint16_t ctc_counter_capture_value_read(void);
<b>Function descriptions</b>	read CTC counter capture value
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	counter capture value (0x0000 - 0xFFFF)

Example:

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

### ctc\_counter\_direction\_read

The description of ctc\_counter\_direction\_read is shown as below:

**Table 3-117. Function ctc\_counter\_direction\_read**

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

Example:

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

## ctc\_counter\_reload\_value\_read

The description of ctc\_counter\_reload\_value\_read is shown as below:

**Table 3-118. Function ctc\_counter\_reload\_value\_read**

<b>Function name</b>	ctc_counter_reload_value_read
<b>Function prototype</b>	uint16_t ctc_counter_reload_value_read(void);
<b>Function descriptions</b>	read CTC counter reload value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	counter reload value (0x0000 - 0xFFFF)

Example:

```
/* read CTC counter reload value */
uint16_t ctc_reload_value = 0;
ctc_reload_value = ctc_counter_reload_value_read ();
```

## ctc\_irc48m\_trim\_value\_read

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

**Table 3-119. Function ctc\_irc48m\_trim\_value\_read**

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



Output parameter{out}	
-	-
Return value	
uint8_t	6-bit IRC48M trim value (0-63)

Example:

```
/* read the IRC48M trim value */
uint8_t ctc_trim_value = 0;
ctc_trim_value = ctc_irc48m_trim_value_read ();
```

### ctc\_interrupt\_enable

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

**Table 3-120. Function ctc\_interrupt\_enable**

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

Example:

```
/* enable CTC clock trim OK interrupt */
```

```
ctc_interrupt_enable (CTC_INT_CKOK);
```

### ctc\_interrupt\_disable

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

**Table 3-121. Function ctc\_interrupt\_disable**

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

Example:

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

### ctc\_interrupt\_flag\_get

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

**Table 3-122. Function ctc\_interrupt\_flag\_get**

<b>Function name</b>	ctc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus ctc_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	get CTC interrupt flag

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	CTC interrupt flag
<i>CTC_INT_FLAG_CKOK</i>	clock trim OK interrupt
<i>CTC_INT_FLAG_CKWARN</i>	clock trim warning interrupt
<i>CTC_INT_FLAG_ERROR</i>	error interrupt
<i>CTC_INT_FLAG_EXPECTREF</i>	expect reference interrupt
<i>CTC_INT_FLAG_CLOCKTRIMERR</i>	clock trim error bit interrupt
<i>CTC_INT_FLAG_REFMISS</i>	reference sync pulse miss interrupt
<i>CTC_INT_FLAG_TRIMERR</i>	trim value error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get CTC interrupt flag status */
```

```
FlagStatus state = ctc_interrupt_flag_get (CTC_INT_FLAG_CKOK);
```

### **ctc\_interrupt\_flag\_clear**

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

**Table 3-123. Function ctc\_interrupt\_flag\_clear**

<b>Function name</b>	ctc_interrupt_flag_clear
<b>Function prototype</b>	void ctc_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear CTC interrupt flag
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	CTC interrupt flag
<i>CTC_INT_FLAG_CKOK</i>	clock trim OK interrupt
<i>CTC_INT_FLAG_CKWARN</i>	clock trim warning interrupt
<i>CTC_INT_FLAG_ERR</i>	error interrupt
<i>CTC_INT_FLAG_EREFP</i>	expect reference interrupt
<i>CTC_INT_FLAG_CKERR</i>	clock trim error bit interrupt
<i>CTC_INT_FLAG_REFMISS</i>	reference sync pulse miss interrupt
<i>CTC_INT_FLAG_TRIMERR</i>	trim value error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
ctc_interrupt_flag_clear (CTC_INT_FLAG_CKOK);
```

### ctc\_flag\_get

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

**Table 3-124. Function ctc\_flag\_get**

<b>Function name</b>	ctc_flag_get
<b>Function prototype</b>	FlagStatus ctc_flag_get(uint32_t flag);
<b>Function descriptions</b>	get CTC status flag
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>flag</b>	CTC status flag
<i>CTC_FLAG_CKOK</i>	clock trim OK interrupt flag
<i>CTC_FLAG_CKWARN</i>	clock trim warning interrupt flag
<i>CTC_FLAG_ERR</i>	error interrupt flag
<i>CTC_FLAG_EREf</i>	expect reference interrupt flag
<i>CTC_FLAG_CKERR</i>	clock trim error bit
<i>CTC_FLAG_REFMISs</i>	reference sync pulse miss flag
<i>CTC_FLAG_TRIMERR</i>	trim value error flag
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get CTC flag status */
```

```
FlagStatus state = ctc_flag_get (CTC_FLAG_CKOK);
```

### ctc\_flag\_clear

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

**Table 3-125. Function ctc\_flag\_clear**

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

<i>CTC_FLAG_ERR</i>	error interrupt flag
<i>CTC_FLAG_ERE</i>	expect reference interrupt flag
<i>CTC_FLAG_CKERR</i>	clock trim error bit
<i>CTC_FLAG_REFMISS</i>	reference sync pulse miss flag
<i>CTC_FLAG_TRIMERR</i>	trim value error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
ctc_flag_clear (CTC_FLAG_CKOK);
```

## 3.7. DAC

The Digital-to-analog converter converts 12-bit digital data to a voltage on the external pins. The DAC registers are listed in chapter [3.7.1](#), the DAC firmware functions are introduced in chapter [3.7.2](#).

### 3.7.1. Peripheral register description

DAC registers are listed in the table shown as below:

**Table 3-126. DAC Registers**

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

Register	Descriptions
DAC_OUT1_DO	DACx_OUT1 data output register

### 3.7.2. Descriptions of Peripheral functions

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

**Table 3-127. DAC firmware functions**

Function name	Function description
<code>dac_deinit</code>	deinitialize DAC
<code>dac_enable</code>	enable DAC
<code>dac_disable</code>	disable DAC
<code>dac_dma_enable</code>	enable DAC DMA function
<code>dac_dma_disable</code>	disable DAC DMA function
<code>dac_output_buffer_enable</code>	enable DAC output buffer
<code>dac_output_buffer_disable</code>	disable DAC output buffer
<code>dac_output_value_get</code>	get DAC output value
<code>dac_data_set</code>	set DAC data holding register value
<code>dac_trigger_enable</code>	enable DAC trigger
<code>dac_trigger_disable</code>	disable DAC trigger
<code>dac_trigger_source_config</code>	configure DAC trigger source
<code>dac_software_trigger_enable</code>	enable DAC software trigger
<code>dac_wave_mode_config</code>	configure DAC wave mode
<code>dac_lfsr_noise_config</code>	configure DAC LFSR noise mode
<code>dac_triangle_noise_config</code>	configure DAC triangle noise mode
<code>dac_concurrent_enable</code>	enable DAC concurrent mode
<code>dac_concurrent_disable</code>	disable DAC concurrent mode
<code>dac_concurrent_software_trigger_enable</code>	enable DAC concurrent software trigger
<code>dac_concurrent_output_buffer_enable</code>	enable DAC concurrent buffer function
<code>dac_concurrent_output_buffer_disable</code>	disable DAC concurrent buffer function
<code>dac_concurrent_data_set</code>	set DAC concurrent mode data holding register value

#### **dac\_deinit**

The description of `dac_deinit` is shown as below:

**Table 3-128. Function `dac_deinit`**

Function name	<code>dac_deinit</code>
Function prototype	<code>void dac_deinit(uint32_t dac_periph);</code>
Function descriptions	deinitialize DAC
Precondition	-
The called functions	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
Input parameter{in}	

<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
dac_deinit(DAC0);
```

### **dac\_enable**

The description of dac\_enable is shown as below:

**Table 3-129. Function dac\_enable**

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

Example:

```
/* enable DAC0_OUT0 */
```

```
dac_enable(DAC0, DAC_OUT0);
```

### **dac\_disable**

The description of dac\_disable is shown as below:

**Table 3-130. Function dac\_disable**

<b>Function name</b>	dac_disable
<b>Function prototype</b>	void dac_disable(uint32_t dac_periph, uint8_t dac_out);



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

Example:

```
/* disable DAC0_OUT0 */
```

```
dac_disable(DAC0, DAC_OUT0);
```

### **dac\_dma\_enable**

The description of `dac_dma_enable` is shown as below:

**Table 3-131. Function `dac_dma_enable`**

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

Example:

```
/* enable DAC0_OUT0 DMA function */
```

```
dac_dma_enable(DAC0, DAC_OUT0);
```

## dac\_dma\_disable

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

**Table 3-132. Function dac\_dma\_disable**

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

Example:

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

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

## dac\_output\_buffer\_enable

The description of dac\_output\_buffer\_enable is shown as below:

**Table 3-133. Function dac\_output\_buffer\_enable**

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

-	-
---	---

Example:

```
/* enable DAC0_OUT0 output buffer */
dac_output_buffer_enable(DAC0, DAC_OUT0);
```

### **dac\_output\_buffer\_disable**

The description of dac\_output\_buffer\_disable is shown as below:

**Table 3-134. Function dac\_output\_buffer\_disable**

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

Example:

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

### **dac\_output\_value\_get**

The description of dac\_output\_value\_get is shown as below:

**Table 3-135. Function dac\_output\_value\_get**

<b>Function name</b>	dac_output_value_get
<b>Function prototype</b>	uint16_t dac_output_value_get(uint32_t dac_periph);
<b>Function descriptions</b>	get DAC output value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral

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

Example:

```
/* get the DAC0_OUT0 last data output value */
```

```
uint16_t data = 0;
```

```
data = dac_output_value_get(DAC0, DAC_OUT0);
```

### **dac\_data\_set**

The description of `dac_data_set` is shown as below:

**Table 3-136. Function `dac_data_set`**

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

Example:

```
/* set DAC0_OUT0 data holding register value */
dac_data_set(DAC0, DAC_OUT0, DAC_ALIGN_8B_R, 0xFF);
```

### **dac\_trigger\_enable**

The description of dac\_trigger\_enable is shown as below:

**Table 3-137. Function dac\_trigger\_enable**

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

Example:

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

### **dac\_trigger\_disable**

The description of dac\_trigger\_disable is shown as below:

**Table 3-138. Function dac\_trigger\_disable**

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

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

Example:

```
/* disable DAC0_OUT0 trigger */
```

```
dac_trigger_disable(DAC0, DAC_OUT0);
```

### **dac\_trigger\_source\_config**

The description of `dac_trigger_source_config` is shown as below:

**Table 3-139. Function `dac_trigger_source_config`**

<b>Function name</b>	<code>dac_trigger_source_config</code>
<b>Function prototype</b>	<code>void dac_trigger_source_config(uint32_t dac_periph, uint8_t dac_out, uint32_t triggersource);</code>
<b>Function descriptions</b>	configure DAC trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
<b>Input parameter{in}</b>	
<b>dac_out</b>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
<b>Input parameter{in}</b>	
<b>triggersource</b>	external trigger of DAC
<i>DAC_TRIGGER_T5_TRGO</i>	TIMER5 TRGO
<i>DAC_TRIGGER_T2_TRGO</i>	TIMER2 TRGO
<i>DAC_TRIGGER_T6_TRGO</i>	TIMER6 TRGO
<i>DAC_TRIGGER_T4_TRGO</i>	TIMER4 TRGO
<i>DAC_TRIGGER_T1_TRGO</i>	TIMER1 TRGO
<i>DAC_TRIGGER_T3_TRGO</i>	TIMER3 TRGO
<i>DAC_TRIGGER_EXTI_9</i>	EXTI interrupt line9 event

<i>DAC_TRIGGER_SOFTWA RE</i>	software trigger
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### **dac\_software\_trigger\_enable**

The description of `dac_trigger_source_enable` is shown as below:

**Table 3-140. Function `dac_software_trigger_enable`**

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

Example:

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

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

### **dac\_wave\_mode\_config**

The description of `dac_wave_mode_config` is shown as below:

**Table 3-141. Function `dac_wave_mode_config`**

<b>Function name</b>	<code>dac_wave_mode_config</code>
----------------------	-----------------------------------

<b>Function prototype</b>	void dac_wave_mode_config(uint32_t dac_periph, uint8_t dac_out, uint32_t wave_mode);
<b>Function descriptions</b>	configure DAC wave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
<b>Input parameter{in}</b>	
<b>dac_out</b>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
<b>Input parameter{in}</b>	
<b>wave_mode</b>	DAC wave mode
<i>DAC_WAVE_DISABLE</i>	wave mode disable
<i>DAC_WAVE_MODE_LFSR</i>	LFSR noise mode
<i>DAC_WAVE_MODE_TRIANGLE</i>	triangle noise mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DAC0_OUT0 wave mode */
```

```
dac_wave_mode_config(DAC0, DAC_OUT0, DAC_WAVE_DISABLE);
```

### **dac\_lfsr\_noise\_config**

The description of dac\_lfsr\_noise\_config is shown as below:

**Table 3-142. Function dac\_lfsr\_noise\_config**

<b>Function name</b>	dac_lfsr_noise_config
<b>Function prototype</b>	void dac_lfsr_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t unmask_bits);
<b>Function descriptions</b>	configure DAC LFSR noise mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
<b>Input parameter{in}</b>	
<b>dac_out</b>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)



Input parameter{in}	
<b>unmask_bits</b>	LFSR noise unmask bits
<i>DAC_LFSR_BIT0</i>	unmask the LFSR bit0
<i>DAC_LFSR_BITSx_0</i>	unmask the LFSR bits [x:0] (x = 1,2,3..11)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DAC0_OUT0 LFSR noise mode */
```

```
dac_lfsr_noise_config(DAC0, DAC_OUT0, DAC_LFSR_BIT0);
```

### **dac\_triangle\_noise\_config**

The description of `dac_triangle_noise_config` is shown as below:

**Table 3-143. Function `dac_triangle_noise_config`**

<b>Function name</b>	<code>dac_triangle_noise_config</code>
<b>Function prototype</b>	<code>void dac_triangle_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t amplitude);</code>
<b>Function descriptions</b>	configure DAC triangle noise mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Input parameter{in}	
<b>dac_out</b>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
Input parameter{in}	
<b>amplitude</b>	the amplitude of the triangle
<i>DAC_TRIANGLE_AMPLITUDE_x</i>	$x = 2^n - 1$ (n = 1..12)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

## dac\_concurrent\_enable

The description of dac\_concurrent\_enable is shown as below:

**Table 3-144. Function dac\_concurrent\_enable**

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

Example:

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

```
dac_concurrent_enable(DAC0);
```

## dac\_concurrent\_disable

The description of dac\_concurrent\_disable is shown as below:

**Table 3-145. Function dac\_concurrent\_disable**

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

Example:

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

```
dac_concurrent_disable(DAC0);
```

## **dac\_concurrent\_software\_trigger\_enable**

The description of `dac_concurrent_software_trigger_enable` is shown as below:

**Table 3-146. Function `dac_concurrent_software_trigger_enable`**

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

Example:

```
/* enable DAC0 concurrent software trigger */
dac_concurrent_software_trigger_enable(DAC0);
```

## **dac\_concurrent\_output\_buffer\_enable**

The description of `dac_concurrent_output_buffer_enable` is shown as below:

**Table 3-147. Function `dac_concurrent_output_buffer_enable`**

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

Example:

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

## dac\_concurrent\_output\_buffer\_disable

The description of dac\_concurrent\_output\_buffer\_disable is shown as below:

**Table 3-148. Function dac\_concurrent\_output\_buffer\_disable**

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

Example:

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

## dac\_concurrent\_data\_set

The description of dac\_concurrent\_data\_set is shown as below:

**Table 3-149. Function dac\_concurrent\_data\_set**

<b>Function name</b>	dac_concurrent_data_set
<b>Function prototype</b>	void dac_concurrent_data_set(uint32_t dac_periph, uint32_t dac_align, uint16_t data0, uint16_t data1);
<b>Function descriptions</b>	set DAC concurrent mode data holding register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b>dac_align</b>	DAC data alignment mode
<i>DAC_ALIGN_12B_R</i>	12-bit right-aligned data
<i>DAC_ALIGN_12B_L</i>	12-bit left-aligned data
<i>DAC_ALIGN_8B_R</i>	8-bit right-aligned data
<b>Input parameter{in}</b>	
<b>data0</b>	DAC_OUT0 data to be loaded (0~4095)
<b>Input parameter{in}</b>	

<b>data1</b>	DAC_OUT1 data to be loaded (0~4095)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## 3.8. DBG

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

### 3.8.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

**Table 3-150. DBG Registers**

Registers	Descriptions
DBG_ID	DBG ID code register
DBG_CTL0	DBG control register 0

### 3.8.2. Descriptions of Peripheral functions

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

**Table 3-151. DBG firmware function**

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

Function name	Function description
dbg_trace_pin_disable	disable trace pin assignment

## Enum dbg\_periph\_enum

Table 3-152. Enum dbg\_periph\_enum

Member name	Function description
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted
DBG_TIMER0_HOLD	hold TIMER0 counter when core is halted
DBG_TIMER1_HOLD	hold TIMER1 counter when core is halted
DBG_TIMER2_HOLD	hold TIMER2 counter when core is halted
DBG_TIMER3_HOLD	hold TIMER3 counter when core is halted
DBG_CAN0_HOLD	debug CAN0 kept when core is halted
DBG_I2C0_HOLD	hold I2C0 smbus when core is halted
DBG_I2C1_HOLD	hold I2C1 smbus when core is halted
DBG_TIMER5_HOLD	hold TIMER5 counter when core is halted
DBG_TIMER6_HOLD	hold TIMER6 counter when core is halted
DBG_TIMER7_HOLD	hold TIMER7 counter when core is halted
DBG_TIMER4_HOLD	hold TIMER4 counter when core is halted
DBG_CAN1_HOLD	debug CAN1 kept when core is halted
DBG_TIMER11_HOLD	hold TIMER11 counter when core is halted
DBG_TIMER12_HOLD	hold TIMER12 counter when core is halted
DBG_TIMER13_HOLD	hold TIMER13 counter when core is halted
DBG_TIMER8_HOLD	hold TIMER8 counter when core is halted
DBG_TIMER9_HOLD	hold TIMER9 counter when core is halted
DBG_TIMER10_HOLD	hold TIMER10 counter when core is halted

## dbg\_deinit

The description of dbg\_deinit is shown as below:

Table 3-153. 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 3-154. Function dbg\_id\_get

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

<b>uint32_t</b>	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 3-155. 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
<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:

```
/* 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 3-156. 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 3-157. 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="#">Enum dbg_periph_enum</a>
<i>DBG_FWDGT_HOLD</i>	debug FWDGT kept when core is halted
<i>DBG_WWDGT_HOLD</i>	debug WWDGT kept when core is halted
<i>DBG_CANx_HOLD</i>	x=0,1, hold CANx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1, hold I2Cx smbus when core is halted
<i>DBG_TIMERx_HOLD</i>	x=0,1,2,3,4,5,6,7,8,9,10,11,12,13, hold TIMERx counter when core is halted
<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 3-158. 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="#">Enum dbg_periph_enum</a>
<i>DBG_FWDGT_HOLD</i>	debug FWDGT kept when core is halted
<i>DBG_WWDGT_HOLD</i>	debug WWDGT kept when core is halted

<i>DBG_CANx_HOLD</i>	x=0,1, hold CANx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1, hold I2Cx smbus when core is halted
<i>DBG_TIMERx_HOLD</i>	x=0,1,2,3,4,5,6,7,8,9,10,11,12,13, hold TIMERx counter when core is halted
<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);
```

### dbg\_trace\_pin\_enable

The description of dbg\_trace\_pin\_enable is shown as below:

**Table 3-159. Function dbg\_trace\_pin\_enable**

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

Example:

```
/* enable trace pin assignment */
```

```
dbg_trace_pin_enable();
```

## dbg\_trace\_pin\_disable

The description of dbg\_trace\_pin\_disable is shown as below:

**Table 3-160. Function dbg\_trace\_pin\_disable**

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

Example:

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

## 3.9. DMA

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

### 3.9.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

**Table 3-161. DMA Registers**

Registers	Descriptions
DMA_INTF	Interrupt flag register

Registers	Descriptions
DMA_INTC	Interrupt flag clear register
DMA_CHxCTL (x=0..6)	Channel x control register
DMA_CHxCNT (x=0..6)	Channel x counter register
DMA_CHxPADDR (x=0..6)	Channel x peripheral base address register
DMA_CHxMADDR (x=0..6)	Channel x memory base address register

### 3.9.2. Descriptions of Peripheral functions

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

**Table 3-162. DMA firmware function**

Function name	Function description
dma_deinit	deinitialize DMA a channel registers
dma_struct_para_init	initialize the parameters of DMA struct with the default values
dma_init	initialize DMA channel
dma_circulation_enable	enable DMA circulation mode
dma_circulation_disable	disable DMA circulation mode
dma_memory_to_memory_enable	enable memory to memory mode
dma_memory_to_memory_disable	disable memory to memory mode
dma_channel_enable	enable DMA channel
dma_channel_disable	disable DMA channel
dma_periph_address_config	set DMA peripheral base address
dma_memory_address_config	set DMA memory base address
dma_transfer_number_config	set the number of remaining data to be transferred by the DMA
dma_transfer_number_get	get the number of remaining data to be transferred by the DMA
dma_priority_config	configure priority level of DMA channel

Function name	Function description
<code>dma_memory_width_config</code>	configure transfer data size of memory
<code>dma_periph_width_config</code>	configure transfer data size of peripheral
<code>dma_memory_increase_enable</code>	enable next address increasement algorithm of memory
<code>dma_memory_increase_disable</code>	disable next address increasement algorithm of memory
<code>dma_periph_increase_enable</code>	enable next address increasement algorithm of peripheral
<code>dma_periph_increase_disable</code>	disable next address increasement algorithm of peripheral
<code>dma_transfer_direction_config</code>	configure the direction of data transfer on the channel
<code>dma_flag_get</code>	check DMA flag is set or not
<code>dma_flag_clear</code>	clear the flag of a DMA channel
<code>dma_interrupt_flag_get</code>	check DMA flag and interrupt enable bit is set or not
<code>dma_interrupt_flag_clear</code>	clear the interrupt flag of a DMA channel
<code>dma_interrupt_enable</code>	enable DMA interrupt
<code>dma_interrupt_disable</code>	disable DMA interrupt

## Structure `dma_parameter_struct`

**Table 3-163. Structure `dma_parameter_struct`**

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

## `dma_deinit`

The description of `dma_deinit` is shown as below:

Table 3-164. Function dma\_deinit

Function name	dma_deinit
Function prototype	void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	deinitialize DMA a channel registers
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0,1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### dma\_struct\_para\_init

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

Table 3-165. Function dma\_struct\_para\_init

Function name	dma_struct_para_init
Function prototype	void dma_struct_para_init(dma_parameter_struct* init_struct);
Function descriptions	initialize the parameters of DMA struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	

<b>*init_struct</b>	a spi_parameter_struct address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## dma\_init

The description of dma\_init is shown as below:

**Table 3-166. Function dma\_init**

<b>Function name</b>	dma_init
<b>Function prototype</b>	void dma_init(uint32_t dma_periph, dma_channel_enum channelx, dma_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0: x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-163. Structure dma_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-



Return value	
-	-

Example:

```

/* DMA0 channel0 initialize */
dma_parameter_struct dma_init_struct;
dma_deinit(DMA0, DMA_CH0);
dma_struct_para_init(&dma_init_struct);

dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_8BIT;
dma_init_struct.number = TRANSFER_NUM;
dma_init_struct.periph_addr = (uint32_t)BANK0_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_8BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA0, DMA_CH0, &dma_init_struct);

```

### **dma\_circulation\_enable**

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

**Table 3-167. Function dma\_circulation\_enable**

Function name	dma_circulation_enable
Function prototype	void dma_circulation_enable(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	enable DMA circulation mode
Precondition	-
The called functions	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..</i>	DMA channel selection

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

Example:

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

### **dma\_circulation\_disable**

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

**Table 3-168. Function dma\_circulation\_disable**

<b>Function name</b>	dma_circulation_disable
<b>Function prototype</b>	void dma_circulation_disable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	disable DMA circulation mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA0 channel0 circulation mode */
```

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

### dma\_memory\_to\_memory\_enable

The description of dma\_memory\_to\_memory\_enable is shown as below:

**Table 3-169. Function dma\_memory\_to\_memory\_enable**

<b>Function name</b>	dma_memory_to_memory_enable
<b>Function prototype</b>	void dma_memory_to_memory_enable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	enable memory to memory mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA0 channel0 memory to memory mode */
```

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

### dma\_memory\_to\_memory\_disable

The description of dma\_memory\_to\_memory\_disable is shown as below:

**Table 3-170. Function dma\_memory\_to\_memory\_disable**

<b>Function name</b>	dma_memory_to_memory_disable
----------------------	------------------------------

<b>Function prototype</b>	void dma_memory_to_memory_disable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	disable memory to memory mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **dma\_channel\_enable**

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

**Table 3-171. Function dma\_channel\_enable**

<b>Function name</b>	dma_channel_enable
<b>Function prototype</b>	void dma_channel_enable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	enable DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **dma\_channel\_disable**

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

**Table 3-172. Function dma\_channel\_disable**

<b>Function name</b>	dma_channel_disable
<b>Function prototype</b>	void dma_channel_disable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	disable DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### **dma\_periph\_address\_config**

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

**Table 3-173. Function dma\_periph\_address\_config**

Function name	dma_periph_address_config
Function prototype	void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
Function descriptions	set DMA peripheral base address
Precondition	-
The called functions	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
Input parameter{in}	
<b>address</b>	peripheral base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

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

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

**Table 3-174. Function dma\_memory\_address\_config**

<b>Function name</b>	dma_memory_address_config
<b>Function prototype</b>	void dma_memory_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
<b>Function descriptions</b>	set DMA memory base address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>address</b>	memory base address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
uint8_t g_destbuf[TRANSFER_NUM];

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

## dma\_transfer\_number\_config

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

**Table 3-175. Function dma\_transfer\_number\_config**

<b>Function name</b>	dma_transfer_number_config
<b>Function prototype</b>	void dma_transfer_number_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t number);
<b>Function descriptions</b>	set the number of remaining data to be transferred by the DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>number</b>	data transfer number
<i>0-0xffff</i>	number
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
#define TRANSFER_NUM                0x400

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

## dma\_transfer\_number\_get

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



Table 3-176. Function dma\_transfer\_number\_get

<b>Function name</b>	dma_transfer_number_get
<b>Function prototype</b>	uint32_t dma_transfer_number_get(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	get the number of remaining data to be transferred by the DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	0-0xffff

Example:

```
uint32_t number = 0;
```

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

### dma\_priority\_config

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

Table 3-177. Function dma\_priority\_config

<b>Function name</b>	dma_priority_config
<b>Function prototype</b>	void dma_priority_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priority);
<b>Function descriptions</b>	configure priority level of DMA channel
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<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:

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

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

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

**Table 3-178. Function dma\_memory\_width\_config**

<b>Function name</b>	dma_memory_width_config
<b>Function prototype</b>	void dma_memory_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t mwidth);
<b>Function descriptions</b>	configure transfer data size of memory

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>mwidth</b>	transfer data width of memory
<i>DMA_MEMORY_WIDTH_8BIT</i>	transfer data width of memory is 8-bit
<i>DMA_MEMORY_WIDTH_16BIT</i>	transfer data width of memory is 16-bit
<i>DMA_MEMORY_WIDTH_32BIT</i>	transfer data width of memory is 32-bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

**Table 3-179. Function dma\_periph\_width\_config**

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

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMA<sub>x</sub>(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CH<sub>x</sub>(DMA0: x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>pwidth</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:

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

### **dma\_memory\_increase\_enable**

The description of dma\_memory\_increase\_enable is shown as below:

**Table 3-180. Function dma\_memory\_increase\_enable**

<b>Function name</b>	dma_memory_increase_enable
<b>Function prototype</b>	void dma_memory_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	enable next address increasement algorithm of memory

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMA<sub>x</sub>(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CH<sub>x</sub>(DMA0: x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### dma\_memory\_increase\_disable

The description of dma\_memory\_increase\_disable is shown as below:

**Table 3-181. Function dma\_memory\_increase\_disable**

<b>Function name</b>	dma_memory_increase_disable
<b>Function prototype</b>	void dma_memory_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	disable next address increasement algorithm of memory
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMA<sub>x</sub>(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel

<i>DMA_CHx</i> (DMA0: x=0..6; DMA1: x=0..4)	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### **dma\_periph\_increase\_enable**

The description of dma\_periph\_increase\_enable is shown as below:

**Table 3-182. Function dma\_periph\_increase\_enable**

<b>Function name</b>	dma_periph_increase_enable
<b>Function prototype</b>	void dma_periph_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	enable next address increasement algorithm of peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx</i> (x=0,1)	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i> (DMA0: x=0..6; DMA1: x=0..4)	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### **dma\_periph\_increase\_disable**

The description of dma\_periph\_increase\_disable is shown as below:

**Table 3-183. Function dma\_periph\_increase\_disable**

<b>Function name</b>	dma_periph_increase_disable
<b>Function prototype</b>	void dma_periph_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	disable next address increasement algorithm of peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

**Table 3-184. Function dma\_transfer\_direction\_config**

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

<b>Function descriptions</b>	configure the direction of data transfer on the channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

**Table 3-185. Function dma\_flag\_get**

<b>Function name</b>	dma_flag_get
<b>Function prototype</b>	FlagStatus dma_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	check DMA flag is set or not



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_FLAG_G</i>	global interrupt flag of channel
<i>DMA_FLAG_FTF</i>	full transfer finish flag of channel
<i>DMA_FLAG_HTF</i>	half transfer finish flag of channel
<i>DMA_FLAG_ERR</i>	error flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
FlagStatus flag = RESET;
```

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

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

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

**Table 3-186. Function dma\_flag\_clear**

<b>Function name</b>	dma_flag_clear
<b>Function prototype</b>	void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	clear the flag of a DMA channel

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_FLAG_G</i>	global interrupt flag of channel
<i>DMA_FLAG_FTF</i>	full transfer finish flag of channel
<i>DMA_FLAG_HTF</i>	half transfer finish flag of channel
<i>DMA_FLAG_ERR</i>	error flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

**Table 3-187. Function dma\_interrupt\_flag\_get**

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

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_INT_FLAG_FTF</i>	full transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_HTF</i>	half transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_ER</i>	error interrupt flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

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

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

**Table 3-188. Function dma\_interrupt\_flag\_clear**

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

<b>Function descriptions</b>	clear the interrupt flag of a DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_INT_FLAG_G</i>	global interrupt flag of channel
<i>DMA_INT_FLAG_FTF</i>	full transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_HTF</i>	half transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_ERR</i>	error interrupt flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

Table 3-189. Function dma\_interrupt\_enable

<b>Function name</b>	dma_interrupt_enable
<b>Function prototype</b>	void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
<b>Function descriptions</b>	enable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>source</b>	DMA interrupt source
<i>DMA_INT_FTF</i>	full transfer finish interrupt of channel
<i>DMA_INT_HTF</i>	half transfer finish interrupt of channel
<i>DMA_INT_ERR</i>	error interrupt of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* DMA0 channel0 interrupt configuration */
dma_interrupt_enable(DMA0, DMA_CH0, DMA_INT_FTF);
```

### **dma\_interrupt\_disable**

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

Table 3-190. Function dma\_interrupt\_disable

<b>Function name</b>	dma_interrupt_disable
<b>Function prototype</b>	void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
<b>Function descriptions</b>	disable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(DMA0:x=0..6; DMA1: x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>source</b>	DMA interrupt source
<i>DMA_INT_FTF</i>	full transfer finish interrupt of channel
<i>DMA_INT_HTF</i>	half transfer finish interrupt of channel
<i>DMA_INT_ERR</i>	error interrupt of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* DMA0 channel0 interrupt configuration */
dma_interrupt_disable(DMA0, DMA_CH0, DMA_INT_FTF);
```

### 3.10. EXMC

The external memory controller EXMC, is used as a translator for MCU to access a variety of external memory. The EXMC registers are listed in chapter [3.10.1](#), the EXMC firmware

functions are introduced in chapter [3.10.2](#).

### 3.10.1. Descriptions of Peripheral registers

EXMC registers are listed in the table shown as below:

**Table 3-191. EXMC Registers**

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

### 3.10.2. Descriptions of Peripheral functions

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

**Table 3-192. EXMC firmware function**

Function name	Function description
exmc_norsram_deinit	deinitialize EXMC NOR/SRAM regionx
exmc_norsram_struct_para_init	initialize exmc_norsram_parameter_struct with the default values
exmc_norsram_init	initialize EXMC NOR/SRAM regionx
exmc_norsram_enable	enable EXMC NOR/PSRAM regionx
exmc_norsram_disable	disable EXMC NOR/PSRAM regionx
exmc_nand_deinit	deinitialize EXMC NAND bankx
exmc_nand_struct_para_init	initialize exmc_norsram_parameter_struct with the default values
exmc_nand_init	initialize EXMC NAND bankx

Function name	Function description
exmc_nand_enable	enable EXMC NAND bankx
exmc_nand_disable	disable EXMC NAND bankx
exmc_pccard_deinit	deinitialize EXMC PC card bank
exmc_pccard_struct_para_init	initialize exmc_pccard_parameter_struct with the default values
exmc_pccard_init	initialize EXMC PC card bank
exmc_pccard_enable	enable EXMC PC card bank
exmc_pccard_disable	disable EXMC PC card bank
exmc_norsram_page_size_config	configure CRAM page size
exmc_nand_ecc_config	enable or disable the EXMC NAND ECC function
exmc_ecc_get	get the EXMC ECC value
exmc_interrupt_enable	enable EXMC interrupt
exmc_interrupt_disable	disable EXMC interrupt
exmc_flag_get	get EXMC flag status
exmc_flag_clear	clear EXMC flag status
exmc_interrupt_flag_get	get EXMC interrupt flag
exmc_interrupt_flag_clear	clear EXMC interrupt flag

### Structure exmc\_norsram\_timing\_parameter\_struct

**Table 3-193. Structure exmc\_norsram\_timing\_parameter\_struct**

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



Member name	Function description
asyn_address_setup_time	configure the data setup time, asynchronous access mode valid

### Structure exmc\_norsram\_parameter\_struct

**Table 3-194. Structure exmc\_norsram\_parameter\_struct**

Member name	Function description
norsram_region	select the region of EXMC NOR/SRAM
write_mode	the write mode, synchronous mode or asynchronous mode
extended_mode	enable or disable the extended mode
asyn_wait	enable or disable the asynchronous wait function
nwait_signal	enable or disable the NWAIT signal while in synchronous burst mode
memory_write	enable or disable the write operation
nwait_config	NWAIT signal configuration
wrap_burst_mode	enable or disable the wrap burst mode
nwait_polarity	specifies the polarity of NWAIT signal from memory
burst_mode	enable or disable the burst mode
databus_width	specifies the databus width of external memory
memory_type	specifies the type of external memory
address_data_mux	specifies whether the data bus and address bus are multiplexed
read_write_timing	timing parameters for read and write if the extended mode is not used or the timing parameters for read if the extended mode is used
write_timing	timing parameters for write when the extended mode is used

### Structure exmc\_nand\_pccard\_timing\_parameter\_struct

**Table 3-195. Structure exmc\_nand\_pccard\_timing\_parameter\_struct**

Member name	Function description
databus_hiztime	configure the databus HiZ time for write operation
holdtime	configure the address hold time(or the data hold time for write operation)
waittime	configure the minimum wait time

Member name	Function description
setuptime	configure the address setup time

### Structure exmc\_nand\_parameter\_struct

**Table 3-196. Structure exmc\_nand\_parameter\_struct**

Member name	Function description
nand_bank	select the bank of NAND
ecc_size	the page size for the ECC calculation
atr_latency	configure the latency of ALE low to RB low
ctr_latency	configure the latency of CLE low to RB low
ecc_logic	enable or disable the ECC calculation logic
databus_width	the NAND flash databus width
wait_feature	enable or disable the wait feature
common_space_timing	the timing parameters for NAND flash common space
attribute_space_timing	the timing parameters for NAND flash attribute space

### Structure exmc\_pccard\_parameter\_struct

**Table 3-197. Structure exmc\_pccard\_parameter\_struct**

Member name	Function description
atr_latency	configure the latency of ALE low to RB low
ctr_latency	configure the latency of CLE low to RB low
wait_feature	enable or disable the wait feature
common_space_timing	the timing parameters for PC card common space
attribute_space_timing	the timing parameters for PC card attribute space
io_space_timing	the timing parameters for PC card IO space

### exmc\_norsram\_deinit

The description of exmc\_norsram\_deinit is shown as below:

Table 3-198. Function `exmc_norsram_deinit`

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

Example:

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

### `exmc_norsram_struct_para_init`

The description of `exmc_norsram_struct_para_init` is shown as below:

Table 3-199. Function `exmc_norsram_struct_para_init`

Function name	<code>exmc_norsram_struct_para_init</code>
Function prototype	<code>void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);</code>
Function descriptions	initialize the struct <code>exmc_norsram_parameter_struct</code>
Precondition	-
The called functions	-
Input parameter{in}	
<code>exmc_norsram_init_struct</code>	Structure for initialization, the structure members can refer to <a href="#">Table 3-194. Structure <code>exmc_norsram_parameter_struct</code></a>

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the struct nor_init_struct */
exmc_norsram_parameter_struct nor_init_struct;
exmc_norsram_struct_para_init (&nor_init_struct);
```

### exmc\_norsram\_init

The description of exmc\_norsram\_init is shown as below:

**Table 3-200. Function exmc\_norsram\_init**

Function name	exmc_norsram_init
Function prototype	void exmc_norsram_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
Function descriptions	initialize EXMC NOR/SRAM region
Precondition	-
The called functions	-
Input parameter{in}	
exmc_norsram_init_struct	Structure for initialization, the structure members can refer to <a href="#">Table 3-194. Structure exmc_norsram_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize EXMC NOR/SRAM bank */
exmc_norsram_parameter_struct lcd_init_struct;
exmc_norsram_timing_parameter_struct lcd_timing_init_struct;
/* configure timing parameter */
```

```

lcd_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;

lcd_timing_init_struct.syn_data_latency = EXMC_DATA_LAT_2_CLK;

lcd_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;

lcd_timing_init_struct.bus_latency = 1;

lcd_timing_init_struct.asyn_data_setup_time = 5;

lcd_timing_init_struct.asyn_address_hold_time = 2;

lcd_timing_init_struct.asyn_address_setup_time = 2;

/* configure EXMC bus parameters */

lcd_init_struct.norsram_region = EXMC_BANK0_NORSRAM_REGION1;

lcd_init_struct.write_mode = EXMC_ASYN_WRITE;

lcd_init_struct.extended_mode = DISABLE;

lcd_init_struct.asyn_wait = DISABLE;

lcd_init_struct.nwait_signal = DISABLE;

lcd_init_struct.memory_write = ENABLE;

lcd_init_struct.nwait_config = EXMC_NWAIT_CONFIG_BEFORE;

lcd_init_struct.wrap_burst_mode = DISABLE;

lcd_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;

lcd_init_struct.burst_mode = DISABLE;

lcd_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;

lcd_init_struct.memory_type = EXMC_MEMORY_TYPE_SRAM;

lcd_init_struct.address_data_mux = DISABLE;

lcd_init_struct.read_write_timing = &lcd_timing_init_struct;

lcd_init_struct.write_timing = &lcd_timing_init_struct;

exmc_norsram_init(&lcd_init_struct);

```

### exmc\_norsram\_enable

The description of exmc\_norsram\_enable is shown as below:

**Table 3-201. Function exmc\_norsram\_enable**

<b>Function name</b>	exmc_norsram_enable
<b>Function prototype</b>	void exmc_norsram_enable(uint32_t exmc_norsram_region);

<b>Function descriptions</b>	enable EXMC NOR/PSRAM region
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_norsram_region</b>	EXMC NOR/SRAM region
<b>EXMC_BANK0_NORSRAM_REGIONx</b>	x=0,1,2,3
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the EXMC NOR/SRAM region1 of bank0 */
exmc_norsram_enable(EXMC_BANK0_NORSRAM_REGION1);
```

### exmc\_norsram\_disable

The description of exmc\_norsram\_disable is shown as below:

**Table 3-202. Function exmc\_norsram\_disable**

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

-	-
Return value	
-	-

Example:

```
/* disable the EXMC NOR/SRAM region1 of bank0 */
exmc_norsram_disable(EXMC_BANK0_NORSRAM_REGION1);
```

### exmc\_nand\_deinit

The description of exmc\_nand\_deinit is shown as below:

**Table 3-203. Function exmc\_nand\_deinit**

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

Example:

```
/* deinitialize EXMC NOR/SRAM bank1 */
exmc_norsram_deinit(EXMC_BANK1_NAND);
```

### exmc\_nand\_struct\_para\_init

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

**Table 3-204. Function exmc\_nand\_struct\_para\_init**

Function name	exmc_nand_struct_para_init
---------------	----------------------------

<b>Function prototype</b>	void exmc_nand_struct_para_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
<b>Function descriptions</b>	initialize the struct exmc_nand_parameter_struct
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_nand_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-196. Structure exmc_nand_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the struct nand_init_struct */
exmc_nand_parameter_struct nand_init_struct;
exmc_nand_struct_para_init (&nand_init_struct);
```

### exmc\_nand\_init

The description of exmc\_nand\_init is shown as below:

**Table 3-205. Function exmc\_nand\_init**

<b>Function name</b>	exmc_nand_init
<b>Function prototype</b>	void exmc_nand_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
<b>Function descriptions</b>	initialize EXMC NAND bank
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_nand_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-196. Structure exmc_nand_parameter_struct</a>
<b>Output parameter{out}</b>	



-	-
Return value	
-	-

Example:

```

exmc_nand_parameter_struct nand_init_struct;

exmc_nand_pccard_timing_parameter_struct nand_timing_init_struct;

/* EXMC configuration */

nand_timing_init_struct.setuptime = 5;

nand_timing_init_struct.waittime = 4;

nand_timing_init_struct.holdtime = 2;

nand_timing_init_struct.databus_hiztime = 2;

nand_init_struct.nand_bank = EXMC_BANK1_NAND;

nand_init_struct.ecc_size = EXMC_ECC_SIZE_2048BYTES;

nand_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;

nand_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;

nand_init_struct.ecc_logic = ENABLE;

nand_init_struct.databus_width = EXMC_NAND_DATABUS_WIDTH_8B;

nand_init_struct.wait_feature = ENABLE;

nand_init_struct.common_space_timing = &nand_timing_init_struct;

nand_init_struct.attribute_space_timing = &nand_timing_init_struct;

exmc_nand_init(&nand_init_struct);

```

### exmc\_nand\_enable

The description of exmc\_nand\_enable is shown as below:

**Table 3-206. Function exmc\_nand\_enable**

Function name	exmc_nand_enable
Function prototype	void exmc_nand_enable(uint32_t exmc_nand_bank);
Function descriptions	enable EXMC NAND bank
Precondition	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_nand_bank</b>	the bank of NAND
<i>EXMC_BANKx_NAND</i>	x=1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable EXMC NAND bank1 */
exmc_nand_enable(EXMC_BANK1_NAND);
```

### exmc\_nand\_disable

The description of exmc\_nand\_disable is shown as below:

**Table 3-207. Function exmc\_nand\_disable**

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

Example:

```
/* disable EXMC NAND bank1 */
```

```
exmc_nand_disable(EXMC_BANK1_NAND);
```

## exmc\_pccard\_deinit

The description of exmc\_pccard\_deinit is shown as below:

**Table 3-208. Function exmc\_pccard\_deinit**

<b>Function name</b>	exmc_pccard_deinit
<b>Function prototype</b>	void exmc_pccard_deinit(void);
<b>Function descriptions</b>	deinitialize EXMC PC card bank
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* deinitialize EXMC PC card bank */
```

```
exmc_pccard_deinit();
```

## exmc\_pccard\_struct\_para\_init

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

**Table 3-209. Function exmc\_pccard\_struct\_para\_init**

<b>Function name</b>	exmc_pccard_struct_para_init
<b>Function prototype</b>	void exmc_pccard_struct_para_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);
<b>Function descriptions</b>	initialize the struct exmc_pccard_parameter_struct
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>exmc_pccard_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-197. Structure exmc_pccard_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* initialize the struct pccard_init_struct */
exmc_pccard_parameter_struct pccard_init_struct;
exmc_pccard_struct_para_init (&pccard_init_struct);

```

### exmc\_pccard\_init

The description of exmc\_pccard\_init is shown as below:

**Table 3-210. Function exmc\_pccard\_init**

<b>Function name</b>	exmc_pccard_init
<b>Function prototype</b>	void exmc_pccard_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);
<b>Function descriptions</b>	initialize EXMC PC card bank
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_pccard_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-197. Structure exmc_pccard_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

exmc_pccard_parameter_struct pccard_init_struct;
exmc_nand_pccard_timing_parameter_struct pccard_timing_init_struct;

```

```

/* EXMC configuration */

pccard_timing_init_struct.setuptime = 5;

pccard_timing_init_struct.waittime = 4;

pccard_timing_init_struct.holdtime = 2;

pccard_timing_init_struct.databus_hiztime = 2;

pccard_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;

pccard_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;

pccard_init_struct.wait_feature = ENABLE;

pccard_init_struct.common_space_timing = & pccard_timing_init_struct;

pccard_init_struct.attribute_space_timing = & pccard_timing_init_struct;

pccard_init_struct.io_space_timing = & pccard_timing_init_struct;

exmc_pccard_init(&pccard_init_struct);

```

### exmc\_pccard\_enable

The description of exmc\_pccard\_enable is shown as below:

**Table 3-211. Function exmc\_pccard\_enable**

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

Example:

```

/* enable EXMC PC card bank */

exmc_pccard_enable();

```

## exmc\_pccard\_disable

The description of exmc\_pccard\_disable is shown as below:

**Table 3-212. Function exmc\_pccard\_disable**

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

Example:

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

## exmc\_norsram\_page\_size\_config

The description of exmc\_norsram\_page\_size\_config is shown as below:

**Table 3-213. Function exmc\_norsram\_page\_size\_config**

<b>Function name</b>	exmc_norsram_page_size_config
<b>Function prototype</b>	void exmc_norsram_page_size_config(uint32_t page_size);
<b>Function descriptions</b>	configure CRAM page size
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>page_size</b>	CRAM page size
<b>EXMC_CRAM_AUTO_</b>	the clock is generated only during synchronous access

<i>SPLIT</i>	
<i>EXMC_CRAM_PAGE_SIZE_128_BYTES</i>	page size is 128 bytes
<i>EXMC_CRAM_PAGE_SIZE_256_BYTES</i>	page size is 256 bytes
<i>EXMC_CRAM_PAGE_SIZE_512_BYTES</i>	page size is 512 bytes
<i>EXMC_CRAM_PAGE_SIZE_1024_BYTES</i>	page size is 1024 bytes
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure CRAM page size */
```

```
exmc_norsram_page_size_config (EXMC_CRAM_PAGE_SIZE_128_BYTES);
```

### exmc\_nand\_ecc\_config

The description of exmc\_nand\_ecc\_config is shown as below:

**Table 3-214. Function exmc\_nand\_ecc\_config**

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

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the EXMC NAND ECC function */
exmc_nand_ecc_config(EXMC_BANK1_NAND, ENABLE);
```

### exmc\_ecc\_get

The description of exmc\_ecc\_get is shown as below:

**Table 3-215. Function exmc\_ecc\_get**

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

Example:

```
/* get the EXMC ECC value */
uint32_t ecc_value;
ecc_value = exmc_ecc_get(EXMC_BANK1_NAND);
```

### exmc\_interrupt\_enable

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



Table 3-216. Function `exmc_interrupt_enable`

<b>Function name</b>	<code>exmc_interrupt_enable</code>
<b>Function prototype</b>	<code>void exmc_interrupt_enable(uint32_t exmc_bank, uint32_t interrupt);</code>
<b>Function descriptions</b>	enable EXMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
<code>EXMC_BANK1_NAND</code>	the NAND bank1
<code>EXMC_BANK2_NAND</code>	the NAND bank2
<code>EXMC_BANK3_PCCARD</code>	the PC Card bank
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which interrupt flag
<code>EXMC_NAND_PCCARD_INT_FLAG_RISE</code>	rising edge interrupt and flag
<code>EXMC_NAND_PCCARD_INT_FLAG_LEVEL</code>	high-level interrupt and flag
<code>EXMC_NAND_PCCARD_INT_FLAG_FALL</code>	falling edge interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **`exmc_interrupt_disable`**

The description of `exmc_interrupt_disable` is shown as below:

Table 3-217. Function `exmc_interrupt_disable`

<b>Function name</b>	<code>exmc_interrupt_disable</code>
<b>Function prototype</b>	<code>void exmc_interrupt_disable(uint32_t exmc_bank, uint32_t interrupt);</code>
<b>Function descriptions</b>	disable EXMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
<code>EXMC_BANK1_NAND</code>	the NAND bank1
<code>EXMC_BANK2_NAND</code>	the NAND bank2
<code>EXMC_BANK3_PCCARD</code>	the PC Card bank
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which interrupt flag
<code>EXMC_NAND_PCCARD_INT_FLAG_RISE</code>	rising edge interrupt and flag
<code>EXMC_NAND_PCCARD_INT_FLAG_LEVEL</code>	high-level interrupt and flag
<code>EXMC_NAND_PCCARD_INT_FLAG_FALL</code>	falling edge interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **exmc\_flag\_get**

The description of `exmc_flag_get` is shown as below:

Table 3-218. Function `exmc_flag_get`

<b>Function name</b>	<code>exmc_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus exmc_flag_get(uint32_t exmc_bank, uint32_t flag);</code>
<b>Function descriptions</b>	get EXMC flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
<code>EXMC_BANK1_NAND</code>	the NAND bank1
<code>EXMC_BANK2_NAND</code>	the NAND bank2
<code>EXMC_BANK3_PCCARD</code>	the PC Card bank
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<code>EXMC_NAND_PCCARD_FLAG_RISE</code>	interrupt rising edge status
<code>EXMC_NAND_PCCARD_FLAG_LEVEL</code>	interrupt high-level status
<code>EXMC_NAND_PCCARD_FLAG_FALL</code>	interrupt falling edge status
<code>EXMC_NAND_PCCARD_FLAG_FIFOE</code>	FIFO empty status
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check rising edge status is set or not*/
```

```
if(RESET != exmc_flag_get(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_FLAG_RISE));
```

## exmc\_flag\_clear

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

**Table 3-219. Function exmc\_flag\_clear**

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

Example:

```
/* clear rising edge status */
```

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

## exmc\_interrupt\_flag\_get

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

**Table 3-220. Function exmc\_interrupt\_flag\_get**

<b>Function name</b>	exmc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus exmc_interrupt_flag_get(uint32_t exmc_bank, uint32_t interrupt);
<b>Function descriptions</b>	get EXMC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCARD	the PC Card bank
RD	
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which interrupt flag
EXMC_NAND_PCCARD_INT_FLAG_RISE	rising edge interrupt and flag
EXMC_NAND_PCCARD_INT_FLAG_LEVEL	high-level interrupt and flag
EXMC_NAND_PCCARD_INT_FLAG_FALL	falling edge interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check rising edge interrupt flag is set or not*/
```

```
if(RESET != exmc_interrupt_flag_get(EXMC_BANK1_NAND,
```

EXMC\_NAND\_PCCARD\_INT\_FLAG\_RISE));

### exmc\_interrupt\_flag\_clear

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

**Table 3-221. Function exmc\_interrupt\_flag\_clear**

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

Example:

```
/* clear rising edge interrupt and flag */
exmc_interrupt_flag_clear(EXMC_BANK1_NAND,
```

EXMC\_NAND\_PCCARD\_INT\_FLAG\_RISE);

## 3.11. EXTI

EXTI is the interrupt/event controller in the MCU. It contains up to 19 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.11.1](#), the EXTI firmware functions are introduced in chapter [3.11.2](#).

### 3.11.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

**Table 3-222. 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

### 3.11.2. Descriptions of Peripheral functions

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

**Table 3-223. 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 3-224. Enum exti\_line\_enum**

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

## Enum exti\_mode\_enum

**Table 3-225. Enum exti\_mode\_enum**

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

## Enum exti\_trig\_type\_enum

**Table 3-226. Enum exti\_trig\_type\_enum**

Member name	Function description
EXTI_TRIG_RISING	EXTI rising edge trigger
EXTI_TRIG_FALLING	EXTI falling edge trigger
EXTI_TRIG_BOTH	EXTI rising and falling edge trigger
EXTI_TRIG_NONE	EXTI without rising or falling edge trigger

## exti\_deinit

The description of exti\_deinit is shown as below:



Table 3-227. 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 3-228. Function exti\_init

Function name	exti_init
Function prototype	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
Function descriptions	initialize EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-224. Enum exti_line_enum</a>
Input parameter{in}	
mode	EXTI mode, refer to <a href="#">Table 3-225. Enum exti_mode_enum</a>
Input parameter{in}	
trig_type	trigger type, refer to <a href="#">Table 3-226. Enum exti_trig_type_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure EXTI_0 */
exti_init(EXTI_0, EXTI_INTERRUPT, EXTI_TRIG_BOTH);
```

## exti\_interrupt\_enable

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

**Table 3-229. 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 3-224. 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 3-230. 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 3-224. 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 3-231. 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
<b>EXTI_x</b>	EXTI line x, refer to <a href="#">Table 3-224. 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 3-232. 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 3-224. 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 3-233. 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 3-224. 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 3-234. Function exti\_software\_interrupt\_disable**

<b>Function name</b>	exti_software_interrupt_disable
<b>Function prototype</b>	void exti_software_interrupt_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable the 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 3-224. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-235. Function exti\_flag\_get**

<b>Function name</b>	exti_flag_get
<b>Function prototype</b>	FlagStatus exti_flag_get(exti_line_enum linex);
<b>Function descriptions</b>	get EXTI line x 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 3-224. 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 flag status */
FlagStatus state = exti_flag_get(EXTI_0);
```

## exti\_flag\_clear

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

**Table 3-236. Function exti\_flag\_clear**

<b>Function name</b>	exti_flag_clear
<b>Function prototype</b>	void exti_flag_clear(exti_line_enum linex);
<b>Function descriptions</b>	clear EXTI line x 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 3-224. 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 3-237. 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 3-224. 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 3-238. 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
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-224. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## 3.12. FMC

There is flash controller and option byte for GD32F403xx series. The FMC registers are listed in chapter [3.12.1](#) the FMC firmware functions are introduced in chapter [3.12.2](#).

### 3.12.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

**Table 3-239. FMC Registers**

Registers	Descriptions
FMC_WS	FMC wait state register
FMC_KEY0	FMC unlock key register 0
FMC_OBKEY	FMC option bytes unlock key register
FMC_STAT0	FMC status register 0
FMC_CTL0	FMC control register 0
FMC_ADDR0	FMC address register 0
FMC_OBSTAT	FMC option bytes status register
FMC_WP	FMC erase/program protection register
FMC_KEY1	FMC unlock key register 1
FMC_STAT1	FMC status register 1
FMC_CTL1	FMC control register 1
FMC_ADDR1	FMC address register 1
FMC_WSEN	FMC wait state enable register
FMC_PID	FMC product ID register

### 3.12.2. Descriptions of Peripheral functions

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

**Table 3-240. FMC firmware function**

Function name	Function description
fmc_wscnt_set	set the FMC wait state counter
fmc_unlock	unlock the main FMC operation

Function name	Function description
fmc_bank0_unlock	unlock the FMC bank0 operation
fmc_bank1_unlock	unlock the FMC bank1 operation
fmc_lock	lock the main FMC operation
fmc_bank0_lock	lock the bank0 FMC operation
fmc_bank1_lock	lock the bank1 FMC operation
fmc_page_erase	FMC erase page
fmc_mass_erase	FMC erase whole chip
fmc_bank0_erase	FMC erase whole bank0
fmc_bank1_erase	FMC erase whole bank1
fmc_word_program	FMC program a word at the corresponding address
fmc_halfword_program	FMC program a half word at the corresponding address
fmc_word_reprogram	FMC reprogram a word at the corresponding address without erasing
ob_unlock	unlock the option byte operation
ob_lock	lock the option byte operation
ob_erase	erase the option byte
ob_write_protection_enable	enable write protect
ob_security_protection_config	configure the option byte security protection
ob_user_write	write the FMC option byte
ob_data_program	program option bytes data
ob_user_get	get the FMC option byte user
ob_data_get	get OB_DATA in register FMC_OBSTAT
ob_write_protection_get	get the FMC option byte write protection
ob_spc_get	get option byte security protection code value
fmc_interrupt_enable	enable FMC interrupt
fmc_interrupt_disable	disable FMC interrupt
fmc_flag_get	check flag is set or not
fmc_flag_clear	clear the FMC flag



Function name	Function description
fmc_interrupt_flag_get	get FMC interrupt flag state
fmc_interrupt_flag_clear	clear FMC interrupt flag state
fmc_bank0_state_get	return the FMC bank0 state
fmc_bank1_state_get	return the FMC bank1 state
fmc_bank0_ready_wait	check FMC bank0 ready or not
fmc_bank1_ready_wait	check FMC bank1 ready or not

### Enum fmc\_state\_enum

Table 3-241. Enum fmc\_state\_enum

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

### Enum fmc\_int\_enum

Table 3-242. Enum fmc\_int\_enum

enum name	enum description
FMC_INT_BANK0_END	enable FMC end of program interrupt
FMC_INT_BANK0_ERR	enable FMC error interrupt
FMC_INT_BANK1_END	enable FMC bank1 end of program interrupt
FMC_INT_BANK1_ERR	enable FMC bank1 error interrupt

## Enum fmc\_flag\_enum

**Table 3-243. Enum fmc\_flag\_enum**

enum name	enum description
FMC_FLAG_BANK0_BUSY	FMC bank0 busy flag
FMC_FLAG_BANK0_PGERR	FMC bank0 operation error flag bit
FMC_FLAG_BANK0_WPERR	FMC bank0 erase/program protection error flag bit
FMC_FLAG_BANK0_END	FMC bank0 end of operation flag bit
FMC_FLAG_OBERR	FMC option bytes read error flag
FMC_FLAG_BANK1_BUSY	FMC bank1 busy flag
FMC_FLAG_BANK1_PGERR	FMC bank1 operation error flag bit
FMC_FLAG_BANK1_WPERR	FMC bank1 erase/program protection error flag bit
FMC_FLAG_BANK1_END	FMC bank1 end of operation flag bit

## Enum fmc\_interrupt\_flag\_enum

**Table 3-244. Enum fmc\_interrupt\_flag\_enum**

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

enum name	enum description
ANK0_WPERR	
FMC_INT_FLAG_B ANK0_END	FMC bank1 end of operation interrupt flag bit

### fmc\_wscnt\_set

The description of fmc\_wscnt\_set is shown as below:

**Table 3-245. Function fmc\_wscnt\_set**

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

Example:

```
/* set the wait state counter value */
```

```
fmc_wscnt_set (WS_WSCNT_1);
```

### fmc\_unlock

The description of fmc\_unlock is shown as below:

**Table 3-246. Function fmc\_unlock**

Function name	fmc_unlock
---------------	------------

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

Example:

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

```
fmc_unlock ( );
```

### fmc\_bank0\_unlock

The description of fmc\_bank0\_unlock is shown as below:

**Table 3-247. Function fmc\_bank0\_unlock**

<b>Function name</b>	fmc_bank0_unlock
<b>Function prototype</b>	void fmc_bank0_unlock(void);
<b>Function descriptions</b>	unlock the FMC bank0 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 main FMC BANK0 operation */
```

```
fmc_bank0_unlock( );
```

### fmc\_bank1\_unlock

The description of fmc\_bank1\_unlock is shown as below:

**Table 3-248. Function fmc\_bank1\_unlock**

<b>Function name</b>	fmc_bank1_unlock
<b>Function prototype</b>	void fmc_bank1_unlock(void);
<b>Function descriptions</b>	unlock the FMC bank1 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 main FMC BANK1 operation */
```

```
fmc_bank1_unlock( );
```

### fmc\_lock

The description of fmc\_lock is shown as below:

**Table 3-249. Function fmc\_lock**

<b>Function name</b>	fmc_lock
<b>Function prototype</b>	void fmc_lock(void);
<b>Function descriptions</b>	lock the main FMC operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

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

```
fmc_lock( );
```

### fmc\_bank0\_lock

The description of fmc\_bank0\_lock is shown as below:

**Table 3-250. Function fmc\_bank0\_lock**

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

Example:

```
/* lock the main FMC BANK0 operation */
```

```
fmc_bank0_lock( );
```

### fmc\_bank1\_lock

The description of fmc\_bank1\_lock is shown as below:

Table 3-251. Function fmc\_bank1\_lock

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

Example:

```
/* lock the main FMC BANK1 operation */
```

```
fmc_bank1_lock( );
```

### fmc\_page\_erase

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

Table 3-252. Function fmc\_page\_erase

Function name	fmc_page_erase
Function prototype	fmc_state_enum fmc_page_erase(uint32_t page_address);
Function descriptions	erase page
Precondition	fmc_unlock
The called functions	fmc_bank0_ready_wait/ fmc_bank1_ready_wait
Input parameter{in}	
page_address	the page address to be erased
Output parameter{out}	
-	-
Return value	

<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>
-----------------------	--

Example:

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

```
fmc_unlock();
```

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

### fmc\_mass\_erase

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

**Table 3-253. Function fmc\_mass\_erase**

<b>Function name</b>	fmc_mass_erase
<b>Function prototype</b>	fmc_state_enum fmc_mass_erase(void );
<b>Function descriptions</b>	erase whole chip
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_bank0_ready_wait/ fmc_bank1_ready_wait
<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="#">Enum fmc_state_enum</a>

Example:

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

```
fmc_unlock();
```

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

### fmc\_bank0\_erase

The description of fmc\_bank0\_erase is shown as below:

**Table 3-254. Function fmc\_bank0\_erase**

<b>Function name</b>	fmc_bank0_erase
<b>Function prototype</b>	fmc_state_enum fmc_bank0_erase (void );



<b>Function descriptions</b>	erase bank0
<b>Precondition</b>	fmc_unlock/fmc_bank0_unlock
<b>The called functions</b>	fmc_bank0_ready_wait
<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="#">Enum fmc_state_enum</a>

Example:

```

/* erase bank0 */

fmc_unlock();

fmc_state_enum state = fmc_bank0_erase ( );

```

### fmc\_bank1\_erase

The description of fmc\_bank1\_erase is shown as below:

**Table 3-255. Function fmc\_bank1\_erase**

<b>Function name</b>	fmc_bank1_erase
<b>Function prototype</b>	fmc_state_enum fmc_bank1_erase (void );
<b>Function descriptions</b>	erase bank1
<b>Precondition</b>	fmc_unlock/fmc_bank1_unlock
<b>The called functions</b>	fmc_bank1_ready_wait
<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="#">Enum fmc_state_enum</a>

Example:

```
/* erase bank1 */
```

```
fmc_unlock();
```

```
fmc_state_enum state = fmc_bank1_erase ( );
```

### fmc\_word\_program

The description of fmc\_word\_program is shown as below:

**Table 3-256. 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
<b>The called functions</b>	fmc_bank0_ready_wait/ fmc_bank1_ready_wait
<b>Input parameter{in}</b>	
<b>address</b>	the address to program
<b>Input parameter{in}</b>	
<b>data</b>	the data to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* program a word at the corresponding address */
```

```
fmc_unlock();
```

```
fmc_page_erase(0x08004000);
```

```
fmc_state_enum state = fmc_word_program (0x08004000, 0xaabbccdd);
```

### fmc\_halfword\_program

The description of fmc\_halfword\_program is shown as below:

**Table 3-257. Function fmc\_halfword\_program**

<b>Function name</b>	fmc_halfword_program
----------------------	----------------------

<b>Function prototype</b>	fmc_state_enum fmc_halfword_program(uint32_t address, uint16_t data);
<b>Function descriptions</b>	program a halfword at the corresponding address
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_bank0_ready_wait/ fmc_bank1_ready_wait
<b>Input parameter{in}</b>	
<b>address</b>	the address to program
<b>Input parameter{in}</b>	
<b>data</b>	the data to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```

/* program a half word at the corresponding address */

fmc_unlock();

fmc_page_erase(0x08004000);

fmc_state_enum state = fmc_halfword_program (0x08004000, 0xaabb);

```

### fmc\_word\_reprogram

The description of fmc\_word\_reprogram is shown as below:

**Table 3-258. Function fmc\_word\_reprogram**

<b>Function name</b>	fmc_word_reprogram
<b>Function prototype</b>	fmc_state_enum fmc_word_reprogram(uint32_t address, uint32_t data);
<b>Function descriptions</b>	program a word at the corresponding address without erasing
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_bank0_ready_wait/ fmc_bank1_ready_wait
<b>Input parameter{in}</b>	
<b>address</b>	the address to program
<b>Input parameter{in}</b>	

<b>data</b>	the data to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* program a word at the corresponding address without erasing*/
```

```
fmc_unlock();
```

```
fmc_state_enum state = fmc_word_reprogram (0x08004000, 0xaabbccdd);
```

## ob\_unlock

The description of ob\_unlock is shown as below:

**Table 3-259. 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>	fmc_unlock
<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 */
```

```
fmc_unlock( );
```

```
ob_unlock ( );
```

## ob\_lock

The description of ob\_lock is shown as below:

**Table 3-260. 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>	fmc_unlock
<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 */
```

```
fmc_unlock( );
```

```
ob_lock ( );
```

## ob\_erase

The description of ob\_erase is shown as below:

**Table 3-261. Function ob\_erase**

<b>Function name</b>	ob_erase
<b>Function prototype</b>	fmc_state_enum ob_erase(void);
<b>Function descriptions</b>	erase the FMC option byte
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_bank0_ready_wait
<b>Input parameter{in}</b>	
-	-

Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* erase the FMC option byte */

fmc_unlock( );

ob_unlock( );

fmc_state_enum state = ob_erase ( );
```

### ob\_write\_protection\_enable

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

**Table 3-262. Function ob\_write\_protection\_enable**

Function name	ob_write_protection_enable
Function prototype	fmc_state_enum ob_write_protection_enable(uint32_t ob_wp);
Function descriptions	enable write protection
Precondition	ob_unlock
The called functions	fmc_bank0_ready_wait
Input parameter{in}	
ob_wp	enable write protection
OB_WPx	write protect specify sector x
OB_WP_ALL	write protect all sector
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* enable write protection */

fmc_unlock( );
```

```
ob_unlock( );
```

```
fmc_state_enum state = ob_write_protection_enable (OB_WP7);
```

### ob\_security\_protection\_config

The description of ob\_security\_protection\_config is shown as below:

**Table 3-263. 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>	fmc_bank0_ready_wait
<b>Input parameter{in}</b>	
<b>ob_spc</b>	specify security protection
<i>FMC_NSPC</i>	no security protection
<i>FMC_USPC</i>	under security protection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* enable security protection */
```

```
ob_unlock( );
```

```
fmc_state_enum state = ob_security_protection_config (FMC_USPC);
```

### ob\_user\_write

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

**Table 3-264. Function ob\_user\_write**

<b>Function name</b>	ob_user_write
<b>Function prototype</b>	fmc_state_enum ob_user_write(uint8_t ob_fwdgt, uint8_t ob_deepsleep, uint8_t ob_stdby, uint8_t ob_boot);

<b>Function descriptions</b>	program the FMC user option byte
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_bank0_ready_wait
<b>Input parameter{in}</b>	
<b>ob_fwdgt</b>	option byte watchdog value
<i>OB_FWDGT_SOFTWARE</i>	software free watchdog
<i>OB_FWDGT_HARDWARE</i>	hardware free watchdog
<b>Input parameter{in}</b>	
<b>ob_deepsleep</b>	option byte deepsleep reset value
<i>OB_DEEPSLEEP_NRESET</i>	no reset when entering deepsleep mode
<i>OB_DEEPSLEEP_RST</i>	generate a reset instead of entering deepsleep mode
<b>Input parameter{in}</b>	
<b>ob_stdby</b>	option byte standby reset value
<i>OB_STDBY_NRESET</i>	no reset when entering standby mode
<i>OB_STDBY_RST</i>	generate a reset instead of entering standby mode
<b>Input parameter{in}</b>	
<b>ob_boot</b>	specifies the option byte boot bank value
<i>OB_BOOT_B0</i>	boot from bank0
<i>OB_BOOT_B1</i>	boot from bank1 or bank0 if bank1 is void
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* configure user option byte */
```

```
fmc_unlock( );
```



```
ob_unlock( );

fmc_state_enum state = ob_user_write(OB_FWDGT_HW,OB_DEEPSLEEP_RST,
OB_STDBY_RST);
```

## ob\_data\_program

The description of ob\_data\_program is shown as below:

**Table 3-265. Function ob\_data\_program**

<b>Function name</b>	ob_data_program
<b>Function prototype</b>	fmc_state_enum ob_data_program(uint32_t address, uint8_t data);
<b>Function descriptions</b>	program option bytes data
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_bank0_ready_wait
<b>Input parameter{in}</b>	
<b>address</b>	the option bytes address to be programmed
<b>Input parameter{in}</b>	
<b>data</b>	the byte to be programmed
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* program option bytes data */

fmc_unlock( );

ob_unlock( );

fmc_state_enum state = ob_data_program (0x1fff804, 0x56);
```

## ob\_user\_get

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

**Table 3-266. Function ob\_user\_get**

<b>Function name</b>	ob_user_get
----------------------	-------------

<b>Function prototype</b>	uint8_t ob_user_get(void);
<b>Function descriptions</b>	get the FMC user option byte
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	the FMC user option byte values(0xF0 – 0xFF)

Example:

```
/* get the FMC user option byte */
```

```
uint8_t user = ob_user_get ( );
```

### ob\_data\_get

The description of ob\_data\_get is shown as below:

**Table 3-267. Function ob\_data\_program**

<b>Function name</b>	ob_data_get
<b>Function prototype</b>	uint16_t ob_data_get(void);
<b>Function descriptions</b>	get the FMC data option byte
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
Uint16_t	the FMC data option byte values(0x0 – 0xFFFF)

Example:

```
/* get the FMC data option byte */
```

```
uint16_t data = ob_data_get ( );
```

### ob\_write\_protection\_get

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

**Table 3-268. Function ob\_write\_protection\_get**

<b>Function name</b>	ob_write_protection_get
<b>Function prototype</b>	uint32_t ob_write_protection_get(void);
<b>Function descriptions</b>	get the FMC option byte write protection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	the FMC write protection option byte value(0x0 – 0xFFFFFFFF)

Example:

```
/* get the FMC option byte write protection */
```

```
uint32_t wp = ob_write_protection_get ( );
```

### ob\_spc\_get

The description of ob\_spc\_get is shown as below:

**Table 3-269. Function ob\_spc\_get**

<b>Function name</b>	ob_spc_get
<b>Function prototype</b>	FlagStatus ob_spc_get(void);
<b>Function descriptions</b>	get the FMC option byte security protection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

```
/* get the FMC option byte security protection */
```

```
FlagStatus spc = ob_spc_get( );
```

### fmc\_interrupt\_enable

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

**Table 3-270. Function fmc\_interrupt\_enable**

<b>Function name</b>	fmc_interrupt_enable
<b>Function prototype</b>	void fmc_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable FMC interrupt
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	-
Input parameter{in}	
<b>interrupt</b>	the FMC interrupt source
<i>FMC_INT_BANK0_EN</i> <i>D</i>	enable FMC bank0 end of program interrupt
<i>FMC_INT_BANK0_ER</i> <i>R</i>	enable FMC bank0 error interrupt
<i>FMC_INT_BANK1_EN</i> <i>D</i>	enable FMC bank1 end of program interrupt
<i>FMC_INT_BANK1_ER</i> <i>R</i>	enable FMC bank1 error interrupt
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable FMC BANK0 end of program interrupt */
```

```
fmc_unlock( );
```

```
fmc_interrupt_enable(FMC_INT_BANK0_END);
```

### fmc\_interrupt\_disable

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

**Table 3-271. 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>	disable FMC interrupt
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the FMC interrupt source
<i>FMC_INT_BANK0_END</i>	FMC bank0 end of program interrupt
<i>FMC_INT_BANK0_ERR</i>	FMC bank0 error interrupt
<i>FMC_INT_BANK1_END</i>	FMC bank1 end of program interrupt
<i>FMC_INT_BANK1_ERR</i>	FMC bank1 error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable FMC BANK0 end of program interrupt */
```

```
fmc_unlock( );
```

```
fmc_interrupt_disable(FMC_INT_BANK0_END);
```

## fmc\_flag\_get

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

**Table 3-272. Function fmc\_flag\_get**

Function name	fmc_flag_get
Function prototype	FlagStatus fmc_flag_get(uint32_t flag);
Function descriptions	check flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
flag	check FMC flag
<i>FMC_FLAG_BANK0_B USY</i>	FMC bank0 busy flag bit
<i>FMC_FLAG_BANK0_P GERR</i>	FMC bank0 operation error flag bit
<i>FMC_FLAG_BANK0_W PERR</i>	FMC bank0 erase/program protection error flag bit
<i>FMC_FLAG_BANK0_E ND</i>	FMC bank0 end of operation flag bit
<i>FMC_FLAG_OBERR</i>	option bytes read error flag bit
<i>FMC_FLAG_BANK1_B USY</i>	FMC bank1 busy flag bit
<i>FMC_FLAG_BANK1_P GERR</i>	FMC bank1 operation error flag bit
<i>FMC_FLAG_BANK1_W PERR</i>	FMC bank1 erase/program protection error flag bit
<i>FMC_FLAG_BANK1_E ND</i>	FMC bank1 end of operation flag bit
Output parameter{out}	
-	-

Return value	
FlagStatus	SET or RESET

Example:

```
/* get FMC bank0 end of operation flag bit */
```

```
FlagStatus flag = fmc_flag_get(FMC_FLAG_BANK0_END);
```

### fmc\_flag\_clear

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

**Table 3-273. Function fmc\_flag\_clear**

Function name	fmc_flag_clear
Function prototype	void fmc_flag_clear(uint32_t flag);
Function descriptions	clear the FMC flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	clear FMC flag
FMC_FLAG_BANK0_P GERR	FMC bank0 operation error flag bit
FMC_FLAG_BANK0_W PERR	FMC bank0 erase/program protection error flag bit
FMC_FLAG_BANK0_E ND	FMC bank0 end of operation flag bit
FMC_FLAG_BANK1_P GERR	FMC bank1 operation error flag bit
FMC_FLAG_BANK1_W PERR	FMC bank1 erase/program protection error flag bit
FMC_FLAG_BANK1_E ND	FMC bank1 end of operation flag bit
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* clear FMC bank0 operation error flag bit */
```

```
FlagStatus flag = fmc_flag_clear(FMC_FLAG_BANK0_PGERR);
```

### fmc\_interrupt\_flag\_get

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

**Table 3-274. Function fmc\_interrupt\_flag\_get**

Function name	fmc_interrupt_flag_get
Function prototype	FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum flag);
Function descriptions	get FMC interrupt flag state
Precondition	-
The called functions	-
Input parameter{in}	
flag	FMC flag
<i>FMC_INT_FLAG_BANK0_PGERR</i>	FMC bank0 operation error interrupt flag bit
<i>FMC_INT_FLAG_BANK0_WPERR</i>	FMC bank0 erase/program protection error interrupt flag bit
<i>FMC_INT_FLAG_BANK0_END</i>	FMC bank0 end of operation interrupt flag bit
<i>FMC_INT_FLAG_BANK1_PGERR</i>	FMC bank1 operation error interrupt flag bit
<i>FMC_INT_FLAG_BANK1_WPERR</i>	FMC bank1 erase/program protection error interrupt flag bit
<i>FMC_INT_FLAG_BANK1_END</i>	FMC bank1 end of operation interrupt flag bit
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET



Example:

```
/* get FMC bank0 operation error interrupt flag bit */
```

```
FlagStatus flag = fmc_interrupt_flag_get (FMC_INT_FLAG_BANK0_PGERR);
```

### fmc\_interrupt\_flag\_clear

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

**Table 3-275. Function fmc\_interrupt\_flag\_clear**

Function name	fmc_interrupt_flag_clear
Function prototype	FlagStatus fmc_interrupt_flag_clear (fmc_interrupt_flag_enum flag);
Function descriptions	clear FMC interrupt flag state
Precondition	-
The called functions	-
Input parameter{in}	
flag	clear FMC flag
<i>FMC_INT_FLAG_BANK0_PGERR</i>	FMC bank0 operation error interrupt flag bit
<i>FMC_INT_FLAG_BANK0_WPERR</i>	FMC bank0 erase/program protection error interrupt flag bit
<i>FMC_INT_FLAG_BANK0_END</i>	FMC bank0 end of operation interrupt flag bit
<i>FMC_INT_FLAG_BANK1_PGERR</i>	FMC bank1 operation error interrupt flag bit
<i>FMC_INT_FLAG_BANK1_WPERR</i>	FMC bank1 erase/program protection error interrupt flag bit
<i>FMC_INT_FLAG_BANK1_END</i>	FMC bank1 end of operation interrupt flag bit
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* clear FMC bank0 operation error interrupt flag bit */
```

```
FlagStatus flag = fmc_interrupt_flag_clear (FMC_INT_FLAG_BANK0_PGERR);
```

### fmc\_bank0\_state\_get

The description of fmc\_bank0\_state\_get is shown as below:

**Table 3-276. Function fmc\_bank0\_state\_get**

Function name	fmc_bank0_state_get
Function prototype	fmc_state_enum fmc_bank0_state_get(void);
Function descriptions	get the FMC bank0 state
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* get the FMC BANK0 state */
```

```
fmc_state_enum state = fmc_bank0_state_get( );
```

### fmc\_bank1\_state\_get

The description of fmc\_bank1\_state\_get is shown as below:

**Table 3-277. Function fmc\_bank1\_state\_get**

Function name	fmc_bank1_state_get
Function prototype	fmc_state_enum fmc_bank1_state_get(void);
Function descriptions	get the FMC bank1 state
Precondition	-
The called functions	-
Input parameter{in}	

-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* get the FMC BANK1 state */
```

```
fmc_state_enum state = fmc_bank1_state_get( );
```

### fmc\_bank0\_ready\_wait

The description of fmc\_bank0\_ready\_wait is shown as below:

**Table 3-278. Function fmc\_bank0\_ready\_wait**

Function name	fmc_bank0_ready_wait
Function prototype	fmc_state_enum fmc_bank0_ready_wait(uint32_t timeout);
Function descriptions	check whether FMC bank0 is ready or not
Precondition	-
The called functions	fmc_bank0_state_get
Input parameter{in}	
timeout	count of loop
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* check whether FMC BANK0 is ready or not */
```

```
fmc_state_enum state = fmc_bank0_ready_wait (0x00001000 );
```

### fmc\_bank1\_ready\_wait

The description of fmc\_bank1\_ready\_wait is shown as below:

Table 3-279. Function `fmc_bank1_ready_wait`

Function name	<code>fmc_bank1_ready_wait</code>
Function prototype	<code>fmc_state_enum fmc_bank1_ready_wait(uint32_t timeout);</code>
Function descriptions	check whether FMC bank1 is ready or not
Precondition	-
The called functions	<code>fmc_bank1_state_get</code>
Input parameter{in}	
timeout	count of loop
Output parameter{out}	
-	-
Return value	
<code>fmc_state_enum</code>	state of FMC, refer to <a href="#">Enum <code>fmc_state_enum</code></a>

Example:

```
/* check whether FMC BANK1 is ready or not */
```

```
fmc_state_enum state = fmc_bank1_ready_wait (0x00001000 );
```

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

### 3.13.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

Table 3-280. FWDGT Registers

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

### 3.13.2. Descriptions of Peripheral functions

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

**Table 3-281. FWDGT firmware function**

Function name	Function description
fwdgt_write_enable	enable write access to FWDGT_PSC and FWDGT_RLD
fwdgt_write_disable	disable write access to FWDGT_PSC and FWDGT_RLD
fwdgt_enable	start the free watchdog timer counter
fwdgt_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 3-282. 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 3-283. 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 3-284. Function fwdgt\_enable**

<b>Function name</b>	fwdgt_enable
<b>Function prototype</b>	void fwdgt_enable(void);
<b>Function descriptions</b>	start the free watchdog timer counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* start the free watchdog timer counter */
```

```
wdgt_enable ( );
```

### wdgt\_counter\_reload

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

**Table 3-285. Function wdgt\_counter\_reload**

Function name	wdgt_counter_reload
Function prototype	void wdgt_counter_reload(void);
Function descriptions	reload the counter of FWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reload FWDGT counter */
```

```
wdgt_counter_reload ( );
```

### wdgt\_config

The description of wdgt\_config is shown as below:

**Table 3-286. Function wdgt\_config**

Function name	wdgt_config
Function prototype	ErrStatus wdgt_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 3-287. Function fwdgt\_flag\_get fwdgt\_write\_disable**

<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);

if(status == RESET)
{
    ...
}else
{
    ...
}

```

## 3.14. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.14.1 Descriptions of Peripheral 4](#), the GPIO firmware functions are introduced in chapter [3.14.2 Descriptions of Peripheral 5](#).

### 3.14.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

**Table 3-288. GPIO Registers**

Registers	Descriptions
GPIOx_CTL0	Port control register 0
GPIOx_CTL1	Port control register 1
GPIOx_ISTAT	Port input status register
GPIOx_OCTL	Port output control register
GPIOx_BOP	Port bit operate register
GPIOx_BC	Port bit clear register
GPIOx_LOCK	Port configuration lock register
GPIOx_SPD	Port bit speed register
AFIO_EC	Event control register
AFIO_PCF0	AFIO port configuration register 0
AFIO_EXTISS0	EXTI sources selection register 0
AFIO_EXTISS1	EXTI sources selection register 1
AFIO_EXTISS2	EXTI sources selection register 2
AFIO_EXTISS3	EXTI sources selection register 3
AFIO_PCF1	AFIO port configuration register 1
AFIO_CPSCCTL	IO compensation control register

### 3.14.2. Descriptions of Peripheral functions

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

**Table 3-289. GPIO firmware function**

Function name	Function description
gpio_deinit	reset GPIO port
gpio_afio_deinit	reset alternate function I/O(AFIO)
gpio_init	GPIO parameter initialization
gpio_bit_set	set GPIO pin
gpio_bit_reset	reset GPIO pin
gpio_bit_write	write data to the specified GPIO pin

Function name	Function description
gpio_port_write	write data to the specified GPIO port
gpio_input_bit_get	get GPIO pin input status
gpio_input_port_get	get GPIO port input status
gpio_output_bit_get	get GPIO pin output status
gpio_output_port_get	get GPIO port output status
gpio_pin_remap_config	configure GPIO pin remap
gpio_exti_source_select	select GPIO pin exti sources
gpio_event_output_config	configure GPIO pin event output
gpio_event_output_enable	enable GPIO pin event output
gpio_event_output_disable	disable GPIO pin event output
gpio_pin_lock	lock GPIO pin
gpio_compensation_config	configure the I/O compensation cell
gpio_compensation_flag_get	check the I/O compensation cell is ready or not

## gpio\_deinit

The description of gpio\_deinit is shown as below:

**Table 3-290. Function gpio\_deinit**

<b>Function name</b>	gpio_deinit
<b>Function prototype</b>	void gpio_deinit(uint32_t gpio_periph);
<b>Function descriptions</b>	reset GPIO port
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x=A,B,C,D,E,F,G)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* reset GPIOA */
```

```
gpio_deinit (GPIOA);
```

### gpio\_afio\_deinit

The description of gpio\_afio\_deinit is shown as below:

**Table 3-291. Function gpio\_afio\_deinit**

<b>Function name</b>	gpio_afio_deinit
<b>Function prototype</b>	void gpio_afio_deinit(void);
<b>Function descriptions</b>	reset alternate function I/O (AFIO)
<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 alternate function */
```

```
gpio_afio_deinit();
```

### gpio\_init

The description of gpio\_init is shown as below:

**Table 3-292. Function gpio\_init**

<b>Function name</b>	gpio_init
<b>Function prototype</b>	void gpio_init(uint32_t gpio_periph, uint32_t mode, uint32_t speed, uint32_t pin);
<b>Function descriptions</b>	GPIO parameter initialization

<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,D,E,F,G)
<b>Input parameter{in}</b>	
<b>gpio_mode</b>	gpio pin mode
<i>GPIO_MODE_AIN</i>	analog input mode
<i>GPIO_MODE_IN_FLOATING</i>	floating input mode
<i>GPIO_MODE_IPD</i>	pull-down input mode
<i>GPIO_MODE_IPU</i>	pull-up input mode
<i>GPIO_MODE_OUT_OD</i>	GPIO output with open-drain
<i>GPIO_MODE_OUT_PP</i>	GPIO output with push-pull
<i>GPIO_MODE_AF_OD</i>	AFIO output with open-drain
<i>GPIO_MODE_AF_PP</i>	AFIO output with push-pull
<b>Input parameter{in}</b>	
<b>speed</b>	gpio output max speed value
<i>GPIO_OSPEED_10MHZ</i>	output max speed 10MHz
<i>GPIO_OSPEED_2MHZ</i>	output max speed 2MHz
<i>GPIO_OSPEED_50MHZ</i>	output max speed 50MHz
<i>GPIO_OSPEED_MAX</i>	output max speed more than 50MHz
<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>	

-	-
Return value	
-	-

Example:

```
/* config PA0 as analog input mode */
```

```
gpio_init(GPIOA, GPIO_MODE_AIN, GPIO_OSPEED_50MHZ, GPIO_PIN_0);
```

### gpio\_bit\_set

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

**Table 3-293. 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
<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,D,E,F,G)
<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 3-294. 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
<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,D,E,F,G)
<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_set (GPIOA, GPIO_PIN_0);
```

## gpio\_bit\_write

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

**Table 3-295. 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,D,E,F,G)
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-296. Function gpio\_port\_write**

<b>Function name</b>	gpio_port_write
<b>Function prototype</b>	void gpio_port_write(uint32_t gpio_periph,uint16_t data);
<b>Function descriptions</b>	write data to the specified GPIO port
<b>Precondition</b>	-
<b>The called functions</b>	-



Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x=A,B,C,D,E,F,G)
Input parameter{in}	
<b>data</b>	specify the value to be written to the port output data register
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### gpio\_input\_bit\_get

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

**Table 3-297. Function gpio\_input\_bit\_get**

<b>Function name</b>	gpio_input_bit_get
<b>Function prototype</b>	FlagStatus gpio_input_bit_get(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	get GPIO pin input status
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x=A,B,C,D,E,F,G)
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}	

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

Example:

```
/* get status of PA0 */
```

```
FlagStatus bit_state;
```

```
bit_state = gpio_input_bit_get (GPIOA, GPIO_PIN_0);
```

### gpio\_input\_port\_get

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

**Table 3-298. 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 port input 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,D,E,F,G)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	0x0000 - 0xFFFF

Example:

```
/* get input value of Port A */
```

```
uint16_t port_state;
```

```
port_state = gpio_input_bit_get(GPIOA);
```

### gpio\_output\_bit\_get

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

Table 3-299. Function gpio\_output\_bit\_get

Function name	gpio_output_bit_get
Function prototype	FlagStatus gpio_output_bit_get(uint32_t gpio_periph,uint32_t pin);
Function descriptions	get GPIO pin output status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x=A,B,C,D,E,F,G)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get output status of PA0 */
```

```
FlagStatus bit_state;
```

```
bit_state = gpio_output_bit_get(GPIOA, GPIO_PIN_0);
```

### gpio\_output\_port\_get

The description of gpio\_output\_port\_get is shown as below:

Table 3-300. Function gpio\_output\_port\_get

Function name	gpio_output_port_get
Function prototype	uint16_t gpio_output_port_get(uint32_t gpio_periph);
Function descriptions	get GPIO port output status
Precondition	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x=A,B,C,D,E,F,G)
<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\_pin\_remap\_config

The description of gpio\_pin\_remap\_config is shown as below:

**Table 3-301. Function gpio\_pin\_remap\_config**

<b>Function name</b>	gpio_pin_remap_config
<b>Function prototype</b>	void gpio_pin_remap_config(uint32_t remap, ControlStatus newvalue);
<b>Function descriptions</b>	configure GPIO pin remap
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>remap</b>	select the pin to remap
<i>GPIO_SPI0_REMAP</i>	SPI0 remapping
<i>GPIO_I2C0_REMAP</i>	I2C0 remapping
<i>GPIO_USART0_REMAP</i>	USART0 remapping
<i>GPIO_USART1_REMAP</i>	USART1 remapping
<i>GPIO_USART2</i>	USART2 partial remapping

<i>_PARTIAL_REMAP</i>	
<i>GPIO_USART2_FULL_REMAP</i>	USART2 full remapping
<i>GPIO_TIMER0_PARTIAL_REMAP</i>	TIMER0 partial remapping
<i>GPIO_TIMER0_FULL_REMAP</i>	TIMER0 full remapping
<i>GPIO_TIMER1_PARTIAL_REMAP</i>	TIMER1 partial remapping
<i>GPIO_TIMER1_FULL_REMAP</i>	TIMER1 full remapping
<i>GPIO_TIMER2_PARTIAL_REMAP</i>	TIMER2 partial remapping
<i>GPIO_TIMER2_FULL_REMAP</i>	TIMER2 full remapping
<i>GPIO_TIMER3_REMAP</i>	TIMER3 remapping
<i>GPIO_CAN0_PARTIAL_REMAP</i>	CAN0 partial remapping
<i>GPIO_CAN0_FULL_REMAP</i>	CAN0 full remapping
<i>GPIO_PD01_REMAP</i>	PD01 remapping
<i>GPIO_TIMER4CH3_IRQ_REMAP</i>	TIMER4 channel3 internal remapping
<i>GPIO_ADC0_ETRGIN_S_REMAP</i>	ADC0 external trigger inserted conversion remapping
<i>GPIO_ADC0_ETRGIN_REG_REMAP</i>	ADC0 external trigger regular conversion remapping
<i>GPIO_ADC1_ETRGIN_S_REMAP</i>	ADC1 external trigger inserted conversion remapping
<i>GPIO_ADC1_ETRGIN_REG_REMAP</i>	ADC1 external trigger regular conversion remapping
<i>GPIO_CAN1_REMAP</i>	CAN1 remapping

<i>GPIO_SWJ_NONJTRST_REMAP</i>	full SWJ(JTAG-DP + SW-DP),but without NJTRST
<i>GPIO_SWJ_SWDPENABLE_REMAP</i>	JTAG-DP disabled and SW-DP enabled
<i>GPIO_SWJ_DISABLE_REMAP</i>	JTAG-DP disabled and SW-DP disabled
<i>GPIO_SPI2_REMAP</i>	SPI2 remapping
<i>GPIO_TIMER8_REMAP</i>	TIMER8 remapping
<i>GPIO_TIMER9_REMAP</i>	TIMER9 remapping
<i>GPIO_TIMER10_REMAP</i>	TIMER10 remapping
<i>GPIO_TIMER12_REMAP</i>	TIMER12 remapping
<i>GPIO_TIMER13_REMAP</i>	TIMER13 remapping
<i>GPIO_EXMC_NADV_REMAP</i>	EXMC_NADV connect/disconnect
<i>GPIO_CTC_REMAP0</i>	CTC remapping (PD15)
<i>GPIO_CTC_REMAP1</i>	CTC remapping (PF0)
<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 SPI0 remapping*/
```

```
gpio_pin_remap_config (GPIO_SPI0_REMAP, ENABLE);
```

### gpio\_exti\_source\_select

The description of gpio\_exti\_source\_select is shown as below:

**Table 3-302. Function gpio\_exti\_source\_select**

<b>Function name</b>	gpio_exti_source_select
<b>Function prototype</b>	void gpio_exti_source_select(uint8_t output_port, uint8_t output_pin);
<b>Function descriptions</b>	select GPIO pin exti sources
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>output_port</b>	gpio event output port
GPIO_PORT_SOURCE_GPIOx	output port source (x=A,B,C,D,E,F,G)
<b>Input parameter{in}</b>	
<b>output_pin</b>	gpio event output pin
GPIO_PIN_SOURCE_x	pin number (x=0..15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config PA0 as EXTI source*/
```

```
gpio_exti_source_select (GPIO_PORT_SOURCE_GPIOA, GPIO_PIN_SOURCE_0);
```

### gpio\_event\_output\_config

The description of gpio\_event\_output\_config is shown as below:

**Table 3-303. Function gpio\_event\_output\_config**

<b>Function name</b>	gpio_event_output_config
<b>Function prototype</b>	void gpio_event_output_config(uint8_t output_port, uint8_t output_pin);

<b>Function descriptions</b>	configure GPIO pin event output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>output_port</b>	gpio event output port
<i>GPIO_EVENT_PORT_GPIOx</i>	event output port x (x= A,B,C,D,E)
<b>Input parameter{in}</b>	
<b>output_pin</b>	gpio event output pin
<i>GPIO_EVENT_PIN_x</i>	pin number (x=0..15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* Config PA0 as the output of event */
```

```
gpio_event_output_config (GPIO_EVENT_PORT_GPIOA, GPIO_EVENT_PIN_0);
```

### gpio\_event\_output\_enable

The description of gpio\_event\_output\_enable is shown as below:

**Table 3-304. Function gpio\_event\_output\_enable**

<b>Function name</b>	gpio_event_output_enable
<b>Function prototype</b>	void gpio_event_output_enable(void);
<b>Function descriptions</b>	enable GPIO pin event output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	



-	-
Return value	
-	-

Example:

```
/* enable GPIO pin event output */
```

```
gpio_event_output_enable(void);
```

### gpio\_event\_output\_disable

The description of gpio\_event\_output\_disable is shown as below:

**Table 3-305. Function gpio\_event\_output\_disable**

Function name	gpio_event_output_disable
Function prototype	void gpio_event_output_disable(void);
Function descriptions	disable GPIO pin event output
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable GPIO pin event output */
```

```
gpio_event_output_disable(void);
```

### gpio\_pin\_lock

The description of gpio\_pin\_lock is shown as below:

**Table 3-306. Function gpio\_pin\_lock**

Function name	gpio_pin_lock
Function prototype	void gpio_pin_lock(uint32_t gpio_periph, uint32_t pin);

<b>Function descriptions</b>	lock 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,D,E,F,G)
<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\_compensation\_config

The description of gpio\_compensation\_config is shown as below:

**Table 3-307. Function gpio\_compensation\_config**

<b>Function name</b>	gpio_compensation_config
<b>Function prototype</b>	void gpio_compensation_config(uint32_t compensation);
<b>Function descriptions</b>	configure the I/O compensation cell
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>compensation</b>	specifies the I/O compensation cell mode
<i>GPIO_COMPENSATIO</i>	I/O compensation cell is enabled

<i>N_ENABLE</i>	
<i>GPIO_COMPENSATION_DISABLE</i>	I/O compensation cell is disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I/O compensation cell */
```

```
gpio_compensation_config (GPIO_COMPENSATION_ENABLE);
```

### gpio\_compensation\_flag\_get

The description of gpio\_compensation\_flag\_get is shown as below:

**Table 3-308. Function gpio\_compensation\_flag\_get**

<b>Function name</b>	gpio_compensation_flag_get
<b>Function prototype</b>	FlagStatus gpio_compensation_flag_get(void);
<b>Function descriptions</b>	check the I/O compensation cell is ready or not
<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:

```
/* check the I/O compensation cell state */
```

```
FlagStatus cell_state;
```

```
cell_state = gpio_compensation_flag_get (void);
```

## 3.15. 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 [3.15.1](#), the I2C firmware functions are introduced in chapter [3.15.2](#).

### 3.15.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

**Table 3-309. I2C Registers**

Registers	Descriptions
I2C_CTL0	Control register 0
I2C_CTL1	Control register 1
I2C_SADDR0	Slave address register 0
I2C_SADDR1	Slave address register 1
I2C_DATA	Transfer buffer register
I2C_STAT0	Transfer status register 0
I2C_STAT1	Transfer status register 1
I2C_CKCFG	Clock configure register
I2C_RT	Rise time register
I2C_FMPCFG	Fast mode plus configure register

### 3.15.2. Descriptions of Peripheral functions

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

**Table 3-310. I2C firmware function**

Function name	Function description
i2c_deinit	reset I2C
i2c_clock_config	configure I2C clock
i2c_mode_addr_config	configure I2C address
i2c_smbus_type_config	SMBus type selection
i2c_ack_config	whether or not to send an ACK

Function name	Function description
i2c_ackpos_config	configure I2C POAP position
i2c_master_addressing	master send slave address
i2c_dualaddr_enable	dual-address mode switch
i2c_enable	enable I2C
i2c_disable	disable I2C
i2c_start_on_bus	generate a START condition on I2C bus
i2c_stop_on_bus	generate a STOP condition on I2C bus
i2c_data_transmit	I2C transmit data function
i2c_data_receive	I2C receive data function
i2c_dma_config	I2C DMA mode enable
i2c_dma_last_transfer_config	configure whether next DMA EOT is DMA last transfer or not
i2c_stretch_scl_low_config	whether to stretch SCL low when data is not ready in slave mode
i2c_slave_response_to_gcall_config	whether or not to response to a general call
i2c_software_reset_config	configure software reset of I2C
i2c_pec_config	configure I2C PEC calculation
i2c_pec_transfer_config	configure whether to transfer PEC value
i2c_pec_value_get	get packet error checking value
i2c_smbus_alert_config	configure I2C alert through SMBA pin
i2c_smbus_arp_config	configure I2C ARP protocol in SMBus
i2c_flag_get	get I2C flag status
i2c_flag_clear	clear I2C flag status
i2c_interrupt_enable	enable I2C interrupt
i2c_interrupt_disable	disable I2C interrupt
i2c_interrupt_flag_get	get I2C interrupt flag status
i2c_interrupt_flag_clear	clear I2C interrupt flag status

## i2c\_deinit

The description of i2c\_deinit is shown as below:

**Table 3-311. 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\_clock\_config

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

**Table 3-312. Function i2c\_clock\_config**

<b>Function name</b>	i2c_clock_config
<b>Function prototype</b>	void i2c_clock_config(uint32_t i2c_periph, uint32_t clkspeed, uint32_t dutyccy);
<b>Function descriptions</b>	I2C clock configure
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	

<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>clkspeed</b>	i2c clock speed
<b>Input parameter{in}</b>	
<b>dutycyc</b>	duty cycle in fast mode
<i>I2C_DTCY_2</i>	T_low/T_high=2
<i>I2C_DTCY_16_9</i>	T_low/T_high=16/9
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2C0 clock speed as 100KHz */
```

```
i2c_clock_config(I2C0, 100000, I2C_DTCY_2);
```

### i2c\_mode\_addr\_config

The description of i2c\_mode\_addr\_config is shown as below:

**Table 3-313. Function i2c\_mode\_addr\_config**

<b>Function name</b>	i2c_mode_addr_config
<b>Function prototype</b>	void i2c_mode_addr_config(uint32_t i2c_periph, uint32_t mode, uint32_t addformat, uint32_t addr);
<b>Function descriptions</b>	configure I2C 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>mode</b>	I2C mode select
<i>I2C_I2CMODE_ENAB</i> <i>LE</i>	I2C mode
<i>I2C_SMBUSMODE_EN</i> <i>ABLE</i>	SMBus mode
<b>Input parameter{in}</b>	
<b>addformat</b>	7bits or 10bits
<i>I2C_ADDFORMAT_7BI</i> <i>TS</i>	7bits
<i>I2C_ADDFORMAT_10</i> <i>BITS</i>	10bits
<b>Input parameter{in}</b>	
<b>addr</b>	I2C address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2C0 address as 0x82, using 7 bits */
```

```
i2c_mode_addr_config(I2C0, I2C_I2CMODE_ENABLE, I2C_ADDFORMAT_7BITS, 0x82);
```

### i2c\_smbus\_type\_config

The description of i2c\_smbus\_type\_config is shown as below:

**Table 3-314. Function i2c\_smbus\_type\_config**

<b>Function name</b>	i2c_smbus_type_config
<b>Function prototype</b>	void i2c_smbus_type_config(uint32_t i2c_periph, uint32_t type);
<b>Function descriptions</b>	SMBus type selection
<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>type</b>	Device or host
<i>I2C_SMBUS_DEVICE</i>	device
<i>I2C_SMBUS_HOST</i>	host
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config I2C0 as SMBUS host type */
```

```
i2c_smbus_type_config (I2C0, I2C_SMBUS_HOST);
```

### **i2c\_ack\_config**

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

**Table 3-315. Function i2c\_ack\_config**

<b>Function name</b>	i2c_ack_config
<b>Function prototype</b>	void i2c_ack_config(uint32_t i2c_periph, uint32_t ack);
<b>Function descriptions</b>	whether or not to send an ACK
<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>ack</b>	I2C peripheral
<i>I2C_ACK_ENABLE</i>	ACK will be sent
<i>I2C_ACK_DISABLE</i>	ACK will not be sent

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 will send ACK */
```

```
i2c_ack_config(I2C0, I2C_ACK_ENABLE);
```

### i2c\_ackpos\_config

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

**Table 3-316. Function i2c\_ackpos\_config**

Function name	i2c_ackpos_config
Function prototype	void i2c_ackpos_config(uint32_t i2c_periph, uint32_t pos);
Function descriptions	I2C POAP position configure
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
pos	ACK position
I2C_ACKPOS_CURRENT	whether to send ACK or not for the current
I2C_ACKPOS_NEXT	whether to send ACK or not for the next byte
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* The ACK of I2C0 is send for the current frame */
```

```
i2c_ackpos_config(I2C0, I2C_ACKPOS_CURRENT);
```

## i2c\_master\_addressing

The description of i2c\_master\_addressing is shown as below:

**Table 3-317. 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 addr, uint32_t trandirection);
<b>Function descriptions</b>	void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection);
<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>addr</b>	slave address
<b>Input parameter{in}</b>	
<b>trandirection</b>	transmitter or receiver
<i>I2C_TRANSMITTER</i>	transmitter
<i>I2C_RECEIVER</i>	receiver
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* send slave address to I2C bus and I2C0 act as receiver */
```

```
i2c_master_addressing(I2C0, 0x82, I2C_RECEIVER);
```

## i2c\_dualaddr\_enable

The description of i2c\_dualaddr\_enable is shown as below:

**Table 3-318. Function i2c\_dualaddr\_enable**

<b>Function name</b>	i2c_dualaddr_enable
<b>Function prototype</b>	void i2c_dualaddr_enable(uint32_t i2c_periph, uint32_t dualaddr);
<b>Function descriptions</b>	dual-address mode switch
<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>dualaddr</b>	Enable or disable
<i>I2C_DUADEN_DISABLE</i>	disable dual-address mode
<i>I2C_DUADEN_ENABLE</i>	enable dual-address mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 dual-address */
```

```
i2c_dualaddr_enable(I2C0, I2C_DUADEN_ENABLE);
```

## i2c\_enable

The description of i2c\_enable is shown as below:

**Table 3-319. Function i2c\_enable**

<b>Function name</b>	i2c_enable
<b>Function prototype</b>	void i2c_enable(uint32_t i2c_periph);

<b>Function descriptions</b>	enable I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### i2c\_disable

The description of i2c\_disable is shown as below:

**Table 3-320. Function i2c\_disable**

<b>Function name</b>	i2c_disable
<b>Function prototype</b>	void i2c_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### i2c\_start\_on\_bus

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

**Table 3-321. Function i2c\_start\_on\_bus**

<b>Function name</b>	i2c_start_on_bus
<b>Function prototype</b>	void i2c_start_on_bus(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a START condition on I2C bus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 send a start condition to I2C bus */
i2c_start_on_bus(I2C0);
```

### i2c\_stop\_on\_bus

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

**Table 3-322. Function i2c\_stop\_on\_bus**

<b>Function name</b>	i2c_stop_on_bus
<b>Function prototype</b>	void i2c_stop_on_bus(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a STOP condition on I2C bus
<b>Precondition</b>	-

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

Example:

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

```
i2c_stop_on_bus(I2C0);
```

### i2c\_data\_transmit

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

**Table 3-323. Function i2c\_data\_transmit**

<b>Function name</b>	i2c_data_transmit
<b>Function prototype</b>	void i2c_data_transmit(uint32_t i2c_periph, uint8_t data);
<b>Function descriptions</b>	I2C transmit data function
<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);
```

### i2c\_data\_receive

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

**Table 3-324. Function i2c\_data\_receive**

<b>Function name</b>	i2c_data_receive
<b>Function prototype</b>	uint8_t i2c_data_receive(uint32_t i2c_periph);
<b>Function descriptions</b>	I2C receive data function
<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>uint8_t</b>	0x00..0xFF

Example:

```
/* I2C0 receive data */
uint8_t i2c_receiver;
i2c_receiver = i2c_data_receive(I2C0);
```

### i2c\_dma\_config

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

**Table 3-325. Function i2c\_dma\_config**

<b>Function name</b>	i2c_dma_config
<b>Function prototype</b>	void i2c_dma_config(uint32_t i2c_periph, uint32_t dmastate);
<b>Function descriptions</b>	enable I2C DMA 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>dmastate</b>	On or off
<i>I2C_DMA_ON</i>	DMA mode enable
<i>I2C_DMA_OFF</i>	DMA mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 DMA mode enable */
```

```
i2c_dma_config(I2C0, I2C_DMA_ON);
```

### **i2c\_dma\_last\_transfer\_config**

The description of i2c\_dma\_last\_transfer\_config is shown as below:

**Table 3-326. Function i2c\_dma\_last\_transfer\_config**

<b>Function name</b>	i2c_dma_last_transfer_config
<b>Function prototype</b>	void i2c_dma_last_transfer_config(uint32_t i2c_periph, uint32_t dmalast);
<b>Function descriptions</b>	configure whether next DMA EOT is DMA last transfer or not
<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>dmalast</b>	next DMA EOT is the last transfer or not
<i>I2C_DMALST_ON</i>	next DMA EOT is the last transfer
<i>I2C_DMALST_OFF</i>	next DMA EOT is not the last transfer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* next DMA EOT is the last transfer */
```

```
i2c_dma_last_transfer_config(I2C0, I2C_DMALST_ON);
```

### **i2c\_stretch\_scl\_low\_config**

The description of i2c\_stretch\_scl\_low\_config is shown as below:

**Table 3-327. Function i2c\_stretch\_scl\_low\_config**

<b>Function name</b>	i2c_stretch_scl_low_config
<b>Function prototype</b>	void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara);
<b>Function descriptions</b>	whether 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>Input parameter{in}</b>	
<b>stretchpara</b>	SCL stretching enable or disable
<i>I2C_SCLSTRETCH_ENABLE</i>	SCL stretching is enabled
<i>I2C_SCLSTRETCH_DISABLE</i>	SCL stretching is disabled
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_config(I2C0, I2C_SCLSTRETCH_ENABLE);
```

### i2c\_slave\_response\_to\_gcall\_config

The description of i2c\_slave\_response\_to\_gcall\_config is shown as below:

**Table 3-328. Function i2c\_slave\_response\_to\_gcall\_config**

Function name	i2c_slave_response_to_gcall_config
Function prototype	void i2c_slave_response_to_gcall_config(uint32_t i2c_periph, uint32_t gcallpara);
Function descriptions	whether or not to response to a general call
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
gcallpara	response to a general call or not
I2C_GCEN_ENABLE	slave will response to a general call
I2C_GCEN_DISABLE	slave will not response to a general call
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 will response to a general call */
```

```
i2c_slave_response_to_gcall_config(I2C0, I2C_GCEN_ENABLE);
```

## i2c\_software\_reset\_config

The description of i2c\_software\_reset\_config is shown as below:

**Table 3-329. Function i2c\_software\_reset\_config**

<b>Function name</b>	i2c_software_reset_config
<b>Function prototype</b>	void i2c_software_reset_config(uint32_t i2c_periph, uint32_t sreset);
<b>Function descriptions</b>	software reset I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>sreset</b>	under reset or not
<i>I2C_SRESET_SET</i>	I2C is under reset
<i>I2C_SRESET_RESET</i>	I2C is not under reset
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
i2c_software_reset_config(I2C0, I2C_SRESET_SET);
```

## i2c\_pec\_config

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

**Table 3-330. Function i2c\_pec\_config**

<b>Function name</b>	i2c_pec_config
<b>Function prototype</b>	void i2c_pec_config(uint32_t i2c_periph, uint32_t pecstate);
<b>Function descriptions</b>	configure I2C PEC calculation

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>pecstate</b>	On or off
<i>I2C_PEC_ENABLE</i>	PEC calculation on
<i>I2C_PEC_DISABLE</i>	PEC calculation off
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* Enable I2C PEC calculation */
```

```
i2c_pec_config(I2C0, I2C_PEC_ENABLE);
```

### **i2c\_pec\_transfer\_config**

The description of i2c\_pec\_transfer\_config is shown as below:

**Table 3-331. Function i2c\_pec\_transfer\_config**

<b>Function name</b>	i2c_pec_transfer_config
<b>Function prototype</b>	void i2c_pec_transfer_config(uint32_t i2c_periph, uint32_t pecpara);
<b>Function descriptions</b>	configure whether to transfer 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)
<b>Input parameter{in}</b>	

<b>pecpara</b>	Transfer PEC or not
<i>I2C_PECTRANS_ENA</i> <i>BLE</i>	transfer PEC
<i>I2C_PECTRANS_DISA</i> <i>BLE</i>	not transfer PEC
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 transfer PEC */
```

```
i2c_pec_transfer_config(I2C0, I2C_PECTRANS_ENABLE);
```

### i2c\_pec\_value\_get

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

**Table 3-332. Function i2c\_pec\_value\_get**

<b>Function name</b>	i2c_pec_value_get
<b>Function prototype</b>	uint8_t i2c_pec_value_get(uint32_t i2c_periph);
<b>Function descriptions</b>	get packet error checking value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	0..255

Example:

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

```
uint8_t pec_value;
```

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

### i2c\_smbus\_alert\_config

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

**Table 3-333. Function i2c\_smbus\_alert\_config**

<b>Function name</b>	i2c_smbus_alert_config
<b>Function prototype</b>	void i2c_smbus_alert_config(uint32_t i2c_periph, uint32_t smbuspara);
<b>Function descriptions</b>	I2C issue alert through SMBA pin
<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>smbuspara</b>	issue alert through SMBA pin or not
<i>I2C_SALTSEND_ENABLE</i>	issue alert through SMBA pin
<i>I2C_SALTSEND_DISABLE</i>	not issue alert through SMBA pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 issue alert through SMBA pin enable */
```

```
i2c_smbus_alert_config(I2C0, I2C_SALTSEND_ENABLE);
```

### i2c\_smbus\_arp\_config

The description of i2c\_smbus\_arp\_config is shown as below:

Table 3-334. Function i2c\_smbus\_arp\_config

Function name	i2c_smbus_arp_config
Function prototype	void i2c_smbus_arp_config(uint32_t i2c_periph, uint32_t arpstate);
Function descriptions	configure I2C ARP protocol in SMBus
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
arpstate	ARP protocol in SMBus switch
I2C_ARP_ENABLE	enable ARP
I2C_ARP_DISABLE	disable ARP
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 ARP protocol in SMBus switch */
i2c_smbus_arp_config(I2C0, I2C_ARP_ENABLE);
```

### i2c\_flag\_get

The description of i2c\_flag\_get is shown as below:

Table 3-335. Function i2c\_flag\_get

Function name	i2c_flag_get
Function prototype	FlagStatus i2c_flag_get(uint32_t i2c_periph, uint32_t flag );
Function descriptions	get I2C flag status
Precondition	-
The called functions	-



Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
<b>flag</b>	specify get which flag
<i>I2C_FLAG_SBSEND</i>	start condition send out
<i>I2C_FLAG_ADDSEND</i>	address is sent in master mode or received and matches in slave mode
<i>I2C_FLAG_BTC</i>	byte transmission finishes
<i>I2C_FLAG_ADD10SEND</i>	header of 10-bit address is sent in master mode
<i>I2C_FLAG_STPDET</i>	stop condition detected in slave mode
<i>I2C_FLAG_RBNE</i>	I2C_DATA is not Empty during receiving
<i>I2C_FLAG_TBE</i>	I2C_DATA is empty during transmitting
<i>I2C_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus
<i>I2C_FLAG_LOSTARB</i>	arbitration lost in master mode
<i>I2C_FLAG_AERR</i>	acknowledge error
<i>I2C_FLAG_OUERR</i>	overflow or underflow situation occurs in slave mode
<i>I2C_FLAG_PECERR</i>	PEC error when receiving data
<i>I2C_FLAG_SMBTO</i>	timeout signal in SMBus mode
<i>I2C_FLAG_SMBALT</i>	SMBus alert status
<i>I2C_FLAG_MASTER</i>	a flag indicating whether I2C block is in master or slave mode
<i>I2C_FLAG_I2CBSY</i>	busy flag
<i>I2C_FLAG_TRS</i>	whether the I2C is a transmitter or a receiver
<i>I2C_FLAG_RXGC</i>	general call address (00h) received
<i>I2C_FLAG_DEFSMB</i>	default address of SMBus device
<i>I2C_FLAG_HSTSMB</i>	SMBus host header detected in slave mode
<i>I2C_FLAG_DUMOD</i>	dual flag in slave mode indicating which address is matched in dual-address mode

<i>I2C_FLAG_TFF</i>	txframe fall flag
<i>I2C_FLAG_TFR</i>	txframe rise flag
<i>I2C_FLAG_RFF</i>	rxframe fall flag
<i>I2C_FLAG_RFR</i>	rxframe rise flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* check whether start condition send out */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get(I2C0, I2C_FLAG_SBSSEND);
```

### **i2c\_flag\_clear**

The description of i2c\_flag\_clear is shown as below:

**Table 3-336. 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>	flag type
<i>I2C_FLAG_SMBALT</i>	SMBus Alert status
<i>I2C_FLAG_SMBTO</i>	timeout signal in SMBus mode
<i>I2C_FLAG_PECERR</i>	PEC error when receiving data

<i>I2C_FLAG_OUERR</i>	over-run or under-run situation occurs in slave mode
<i>I2C_FLAG_AERR</i>	acknowledge error
<i>I2C_FLAG_LOSTARB</i>	arbitration lost in master mode
<i>I2C_FLAG_BERR</i>	a bus error
<i>I2C_FLAG_ADDSEND</i>	cleared by reading I2C_STAT0 and reading I2C_STAT1
<i>I2C_FLAG_TFF</i>	txframe fall flag
<i>I2C_FLAG_TFR</i>	txframe rise flag
<i>I2C_FLAG_RFF</i>	rxframe fall flag
<i>I2C_FLAG_RFR</i>	rxframe rise flag
<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 3-337. 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 inttype);
<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>inttype</b>	interrupt type
<i>I2C_INT_ERR</i>	error interrupt enable
<i>I2C_INT_EV</i>	event interrupt enable
<i>I2C_INT_BUF</i>	<i>buffer interrupt enable</i>
<i>I2C_INT_TFF</i>	<i>txframe fall interrupt enable</i>
<i>I2C_INT_TFR</i>	<i>txframe rise interrupt enable</i>
<i>I2C_INT_RFF</i>	<i>rxframe fall interrupt enable</i>
<i>I2C_INT_RFR</i>	<i>rxframe rise interrupt enable</i>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 error interrupt */
```

```
i2c_interrupt_enable(I2C0, I2C_INT_EV);
```

### i2c\_interrupt\_disable

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

**Table 3-338. 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 inttype);
<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>inttype</b>	interrupt type

<i>I2C_INT_ERR</i>	error interrupt disable
<i>I2C_INT_EV</i>	event interrupt disable
<i>I2C_INT_BUF</i>	buffer interrupt disable
<i>I2C_INT_TFF</i>	<i>txframe fall interrupt enable</i>
<i>I2C_INT_TFR</i>	<i>txframe rise interrupt enable</i>
<i>I2C_INT_RFF</i>	<i>rxframe fall interrupt enable</i>
<i>I2C_INT_RFR</i>	<i>rxframe rise interrupt enable</i>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 error interrupt */
```

```
i2c_interrupt_disable(I2C0, I2C_INT_EV);
```

### **i2c\_interrupt\_flag\_get**

The description of i2c\_interrupt\_flag\_get is shown as below:

**Table 3-339. 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, uint32_t intflag);
<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>intflag</b>	interrupt flag
<i>I2C_INT_FLAG_SBSE</i>	start condition sent out in master mode interrupt flag

<i>ND</i>	
<i>I2C_INT_FLAG_ADDS</i> <i>END</i>	address is sent in master mode or received and matches in slave mode interrupt flag
<i>I2C_INT_FLAG_BTC</i>	byte transmission finishes
<i>I2C_INT_FLAG_ADD10</i> <i>SEND</i>	header of 10-bit address is sent in master mode interrupt flag
<i>I2C_INT_FLAG_STPD</i> <i>ET</i>	stop condition detected in slave mode interrupt flag
<i>I2C_INT_FLAG_RBNE</i>	I2C_DATA is not Empty during receiving interrupt flag
<i>I2C_INT_FLAG_TBE</i>	I2C_DATA is empty during transmitting interrupt flag
<i>I2C_INT_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
<i>I2C_INT_FLAG_LOSTA</i> <i>RB</i>	arbitration lost in master mode interrupt flag
<i>I2C_INT_FLAG_AERR</i>	acknowledge error interrupt flag
<i>I2C_INT_FLAG_OUER</i> <i>R</i>	over-run or under-run situation occurs in slave mode interrupt flag
<i>I2C_INT_FLAG_PECER</i> <i>RR</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBT</i> <i>O</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBA</i> <i>LT</i>	SMBus Alert status interrupt flag
<i>I2C_INT_FLAG_TFF</i>	txframe fall interrupt flag
<i>I2C_INT_FLAG_TFR</i>	txframe rise interrupt flag
<i>I2C_INT_FLAG_RFF</i>	rxframe fall interrupt flag
<i>I2C_INT_FLAG_RFR</i>	rxframe rise interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* check the byte transmission finishes interrupt flag is set or not */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get(I2C0, I2C_INT_FLAG_BTC);
```

### i2c\_interrupt\_flag\_clear

The description of i2c\_interrupt\_flag\_clear is shown as below:

**Table 3-340. Function i2c\_interrupt\_flag\_clear**

Function name	i2c_interrupt_flag_clear
Function prototype	void i2c_interrupt_flag_clear(uint32_t i2c_periph, uint32_t intflag);
Function descriptions	clear I2C interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
<b>intflag</b>	interrupt flag
<i>I2C_INT_FLAG_ADDS END</i>	address is sent in master mode or received and matches in slave mode interrupt flag
<i>I2C_INT_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
<i>I2C_INT_FLAG_LOSTA RB</i>	arbitration lost in master mode interrupt flag
<i>I2C_INT_FLAG_AERR</i>	acknowledge error interrupt flag
<i>I2C_INT_FLAG_OUER R</i>	over-run or under-run situation occurs in slave mode interrupt flag
<i>I2C_INT_FLAG_PECER RR</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBT O</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBA LT</i>	SMBus Alert status interrupt flag

<i>I2C_INT_FLAG_TFF</i>	txframe fall interrupt flag
<i>I2C_INT_FLAG_TFR</i>	txframe rise interrupt flag
<i>I2C_INT_FLAG_RFF</i>	rxframe fall interrupt flag
<i>I2C_INT_FLAG_RFR</i>	rxframe rise interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the acknowledge error interrupt flag */
```

```
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_AERR);
```

## 3.16. MISC

MISC is a software package that provide the interfaces for NVIC and SysTick. The NVIC and SysTick registers are listed in chapter [3.16.1](#), the MISC firmware functions are introduced in chapter [3.16.2](#).

### 3.16.1. Descriptions of Peripheral registers

**Table 3-341. NVIC Registers**

Registers	Descriptions
ISER <sup>(1)</sup>	Interrupt Set Enable Register
ICER <sup>(1)</sup>	Interrupt Clear Enable Register
ISPR <sup>(1)</sup>	Interrupt Set Pending Register
ICPR <sup>(1)</sup>	Interrupt Clear Pending Register
IABR <sup>(1)</sup>	Interrupt Active bit Register
IP <sup>(1)</sup>	Interrupt Priority Register
STIR <sup>(1)</sup>	Software Trigger Interrupt Register
CPUID <sup>(2)</sup>	CPUID Base Register
ICSR <sup>(2)</sup>	Interrupt Control and State Register
VTOR <sup>(2)</sup>	Vector Table Offset Register
AIRCR <sup>(2)</sup>	Application Interrupt and Reset Control Register
SCR <sup>(2)</sup>	System Control Register
CCR <sup>(2)</sup>	Configuration Control Register
SHP <sup>(2)</sup>	System Handlers Priority Registers



Registers	Descriptions
SHCSR <sup>(2)</sup>	System Handler Control and State Register
CFSR <sup>(2)</sup>	Configurable Fault Status Register
HFSR <sup>(2)</sup>	HardFault Status Register
DFSR <sup>(2)</sup>	Debug Fault Status Register
MMFAR <sup>(2)</sup>	MemManage Fault Address Register
BFAR <sup>(2)</sup>	BusFault Address Register
AFSR <sup>(2)</sup>	Auxiliary Fault Status Register
PFR <sup>(2)</sup>	Processor Feature Register
DFR <sup>(2)</sup>	Debug Feature Register
ADR <sup>(2)</sup>	Auxiliary Feature Register
MMFR <sup>(2)</sup>	Memory Model Feature Register
ISAR <sup>(2)</sup>	Instruction Set Attributes Register
CPACR <sup>(2)</sup>	Coprocessor Access Control Register

1. refer to the structure NVIC\_Type, is defined in the core\_cm4.h file

2. refer to the structure SCB\_Type, is defined in the core\_cm4.h file

**Table 3-342. SysTick Registers**

Registers	Descriptions
CTRL <sup>(1)</sup>	SysTick Control and Status Register
LOAD <sup>(1)</sup>	SysTick Reload Value Register
VAL <sup>(1)</sup>	SysTick Current Value Register
CALIB <sup>(1)</sup>	SysTick Calibration Register

1. refer to the structure SysTick\_Type, is defined in the core\_cm4.h file

### 3.16.2. Descriptions of Peripheral functions

#### Enum IRQn\_Type

**Table3-343. IRQn\_Type**

Member name	Function description
WWDGT_IRQn	window watchDog timer interrupt
LVD_IRQn	LVD through EXTI line detect interrupt
TAMPER_IRQn	tamper through EXTI line detect
RTC_IRQn	RTC through EXTI line interrupt
FMC_IRQn	FMC interrupt
RCU_CTC_IRQn	RCU and CTC interrupt
EXTI0_IRQn	EXTI line 0 interrupts
EXTI1_IRQn	EXTI line 1 interrupts
EXTI2_IRQn	EXTI line 2 interrupts
EXTI3_IRQn	EXTI line 3 interrupts
EXTI4_IRQn	EXTI line 4 interrupts

Member name	Function description
DMA0_Channel0_IRQn	DMA0 channel0 interrupt
DMA0_Channel1_IRQn	DMA0 channel1 interrupt
DMA0_Channel2_IRQn	DMA0 channel2 interrupt
DMA0_Channel3_IRQn	DMA0 channel3 interrupt
DMA0_Channel4_IRQn	DMA0 channel4 interrupt
DMA0_Channel5_IRQn	DMA0 channel5 interrupt
DMA0_Channel6_IRQn	DMA0 channel6 interrupt
ADC0_1_IRQn	ADC0 and ADC1 interrupt
CAN0_TX_IRQn	CAN0 transmit interrupts
CAN0_RX0_IRQn	CAN0 receive0 interrupts
CAN0_RX1_IRQn	CAN0 receive1 interrupts
CAN0_EWMC_IRQn	CAN0 EWMC interrupts
EXTI5_9_IRQn	EXTI[9:5] interrupts
TIMER0_BRK_TIMER8_IRQn	TIMER0 break and TIMER8 interrupts
TIMER0_UP_TIMER9_IRQn	TIMER0 update and TIMER9 interrupts
TIMER0_TRG_CMT_TIMER10_IRQn	TIMER0 trigger and commutation and TIMER10 interrupts
TIMER0_Channel_IRQn	TIMER0 channel capture compare interrupt
TIMER1_IRQn	TIMER1 interrupt
TIMER2_IRQn	TIMER2 interrupt
TIMER3_IRQn	TIMER3 interrupt
I2C0_EV_IRQn	I2C0 event interrupt
I2C0_ER_IRQn	I2C0 error interrupt
I2C1_EV_IRQn	I2C1 event interrupt
I2C1_ER_IRQn	I2C1 error interrupt
SPI0_IRQn	SPI0 interrupt
SPI1_IRQn	SPI1 interrupt
USART0_IRQn	USART0 interrupt
USART1_IRQn	USART1 interrupt
USART2_IRQn	USART2 interrupt
EXTI10_15_IRQn	EXTI[15:10] interrupts
RTC_ALARM_IRQn	RTC alarm interrupt
USBFS_WKUP_IRQn	USBFS wakeup interrupt
TIMER7_BRK_TIMER11_IRQn	TIMER7 break and TIMER11 interrupts
TIMER7_UP_TIMER12_IRQn	TIMER7 update and TIMER12 interrupts
TIMER7_TRG_CMT_TIMER13_IRQn	TIMER7 trigger and commutation and TIMER13 interrupts
TIMER7_Channel_IRQn	TIMER7 channel capture compare interrupt
ADC2_IRQn	ADC 2 interrupt
EXMC_IRQn	EXMC global interrupt
SDIO_IRQn	SDIO global interrupt
TIMER4 global interrupt	TIMER4 global interrupt
SPI2_IRQn	SPI2 global interrupt

Member name	Function description
UART3_IRQn	UART3 global interrupt
UART4_IRQn	UART4 global interrupt
TIMER5_IRQn	TIMER5 global interrupt
TIMER6_IRQn	TIMER6 global interrupt
DMA1_Channel0_IRQn	DMA1 channel0 global interrupt
DMA1_Channel1_IRQn	DMA1 channel1 global interrupt
DMA1_Channel2_IRQn	DMA1 channel2 global interrupt
DMA1_Channel3_IRQn	DMA1 channel3 global interrupt
DMA1_Channel4_IRQn	DMA1 channel4 global interrupt
CAN1_TX_IRQn	CAN1 transmit interrupt
CAN1_RX0_IRQn	CAN1 receive0 interrupt
CAN1_RX1_IRQn	CAN1 receive1 interrupt
CAN1_EWMC_IRQn	CAN1 EWMC interrupt
USBFS_IRQn	USBFS global interrupt

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

**Table 3-344. MISC firmware function**

Function name	Function description
nvic_priority_group_set	set the priority group
nvic_irq_enable	enable NVIC interrupt request
nvic_irq_disable	disable NVIC interrupt request
nvic_vector_table_set	set the NVIC vector table base address
system_lowpower_set	set the state of the low power mode
system_lowpower_reset	reset the state of the low power mode
systick_clksource_set	set the systick clock source

## nvic\_priority\_group\_set

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

**Table 3-345. Function nvic\_priority\_group\_set**

<b>Function name</b>	nvic_priority_group_set
<b>Function prototype</b>	void nvic_priority_group_set(uint32_t nvic_prigroup);
<b>Function descriptions</b>	configure bits length of the priority group
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>nvic_prigroup</b>	priority group
<i>NVIC_PRIGROUP_PR E0_SUB4</i>	0 bits for pre-emption priority 4 bits for subpriority
<i>NVIC_PRIGROUP_PR E1_SUB3</i>	1 bits for pre-emption priority 3 bits for subpriority

<i>NVIC_PRIGROUP_PRE2_SUB2</i>	2 bits for pre-emption priority 2 bits for subpriority
<i>NVIC_PRIGROUP_PRE3_SUB1</i>	3 bits for pre-emption priority 1 bits for subpriority
<i>NVIC_PRIGROUP_PRE4_SUB0</i>	4 bits for pre-emption priority 0 bits for subpriority
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* priority group configuration , 0 bits for pre-emption priority 4 bits for subpriority */
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

### **nvic\_irq\_enable**

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

**Table 3-346. Function nvic\_irq\_enable**

<b>Function name</b>	nvic_irq_enable
<b>Function prototype</b>	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
<b>Function descriptions</b>	enable NVIC interrupt request
<b>Precondition</b>	-
<b>The called functions</b>	nvic_priority_group_set
<b>Input parameter{in}</b>	
<b>nvic_irq</b>	NVIC interrupt, refer to enum <a href="#">Enum IRQn_Type</a>
<b>Input parameter{in}</b>	
<b>nvic_irq_pre_priority</b>	the pre-emption priority needed to set
<b>Input parameter{in}</b>	
<b>nvic_irq_sub_priority</b>	the subpriority needed to set
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable window watchDog timer interrupt , pre-emption priority is 1, subpriority is 1 */
nvic_irq_enable(WWDGT_IRQn, 1, 1);
```

### **nvic\_irq\_disable**

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

Table 3-347. Function nvic\_irq\_disable

Function name	nvic_irq_disable
Function prototype	void nvic_irq_disable(uint8_t nvic_irq);
Function descriptions	disable NVIC interrupt request
Precondition	-
The called functions	-
Input parameter{in}	
nvic_irq	NVIC interrupt, refer to enum <a href="#">Enum IRQn_Type</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable window watchDog timer interrupt */
nvic_irq_disable(WWDGT_IRQn);
```

### nvic\_vector\_table\_set

The description of nvic\_vector\_table\_set is shown as below:

Table 3-348. Function nvic\_vector\_table\_set

Function name	nvic_vector_table_set
Function prototype	void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);
Function descriptions	set the NVIC vector table base address
Precondition	-
The called functions	-
Input parameter{in}	
nvic_vect_tab	the RAM or FLASH base address
NVIC_VECTTAB_RAM	RAM base address
NVIC_VECTTAB_FLASH	Flash base address
Input parameter{in}	
offset	vector table offset (vector table start address= base address+offset)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set vector table address = NVIC_VECTTAB_FLASH + 0x200 */
nvic_vector_table_set (NVIC_VECTTAB_FLASH, 0x200);
```

## system\_lowpower\_set

The description of system\_lowpower\_set is shown as below:

**Table 3-349. Function system\_lowpower\_set**

<b>Function name</b>	system_lowpower_set
<b>Function prototype</b>	void system_lowpower_set(uint8_t lowpower_mode);
<b>Function descriptions</b>	set the state of the low power mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowpower_mode</b>	the low power mode state
SCB_LPM_SLEEP_EXIT_ISR	if chose this para, the system always enter low power mode by exiting from ISR
SCB_LPM_DEEPSLEEP	if chose this para, the system will enter the DEEPSLEEP mode
SCB_LPM_WAKE_BY_ALL_INT	if chose this para, the lowpower mode can be woke up by all the enable and disable interrupts
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* the system always enter low power mode by exiting from ISR */
system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);
```

## system\_lowpower\_reset

The description of system\_lowpower\_reset is shown as below:

**Table 3-350. Function system\_lowpower\_reset**

<b>Function name</b>	system_lowpower_reset
<b>Function prototype</b>	void system_lowpower_reset(uint8_t lowpower_mode);
<b>Function descriptions</b>	reset the state of the low power mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowpower_mode</b>	the low power mode state
SCB_LPM_SLEEP_EXIT_ISR	if chose this para, the system will exit low power mode by exiting from ISR
SCB_LPM_DEEPSLEEP	if chose this para, the system will enter the SLEEP mode
SCB_LPM_WAKE_BY_ALL_INT	if chose this para, the lowpower mode only can be woke up by the enable

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

Example:

```
/* the system will exit low power mode by exiting from ISR */
system_lowpower_reset (SCB_LPM_SLEEP_EXIT_ISR);
```

### systick\_clksource\_set

The description of systick\_clksource\_set is shown as below:

**Table 3-351. Function systick\_clksource\_set**

<b>Function name</b>	systick_clksource_set
<b>Function prototype</b>	void systick_clksource_set(uint32_t systick_clksource);
<b>Function descriptions</b>	set the systick clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>systick_clksource</b>	the systick clock source needed to choose
SYSTICK_CLKSOURC E_HCLK	systick clock source is from HCLK
SYSTICK_CLKSOURC E_HCLK_DIV8	systick clock source is from HCLK/8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* systick clock source is HCLK/8 */
systick_clksource_set (SYSTICK_CLKSOURCE_HCLK_DIV8);
```

## 3.17. PMU

According to the Power management unit (PMU), provides three types of power saving modes, including Sleep, Deep-sleep and Standby mode. The PMU registers are listed in chapter [3.17.1](#), the PMU firmware functions are introduced in chapter [3.17.2](#).

### 3.17.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

**Table 3-352. PMU Registers**

Registers	Descriptions
PMU_CTL	Control register
PMU_CS	Control and status register

### 3.17.2. Descriptions of Peripheral functions

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

**Table 3-353. PMU firmware function**

Function name	Function description
pmu_deinit	deinitialize the PMU
pmu_lvd_select	select low voltage detector threshold
pmu_ldo_output_select	select LDO output voltage
pmu_lvd_disable	disable PMU lvd
pmu_highdriver_switch_select	switch high-driver mode
pmu_highdriver_mode_enable	enable high-driver mode
pmu_highdriver_mode_disable	disable high-driver mode
pmu_lowdriver_mode_enable	enable low-driver mode in deep-sleep mode
pmu_lowdriver_mode_disable	disable low-driver mode in deep-sleep mode
pmu_lowpower_driver_config	in deep-sleep mode, driver mode when use low power LDO
pmu_normalpower_driver_config	in deep-sleep mode, driver mode when use normal power LDO
pmu_to_sleepmode	PMU work at sleep mode
pmu_to_deepsleepmode	PMU work at deepsleep mode
pmu_to_standbymode	pmu work at standby mode
pmu_wakeup_pin_enable	enable wakeup pin
pmu_wakeup_pin_disable	disable wakeup pin
pmu_backup_write_enable	enable backup domain write



Function name	Function description
pmu_backup_write_disable	disable backup domain write
pmu_flag_clear	clear flag bit
pmu_flag_get	get flag state

### pmu\_deinit

The description of pmu\_deinit is shown as below:

**Table 3-354. Function pmu\_deinit**

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

Example:

```
/* reset PMU */
```

```
pmu_deinit ();
```

### pmu\_lvd\_select

The description of pmu\_lvd\_select is shown as below:

**Table 3-355. Function pmu\_lvd\_select**

Function name	pmu_lvd_select
Function prototype	void pmu_lvd_select(uint32_t lvd_t_n);
Function descriptions	select low voltage detector threshold
Precondition	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lvdt_n</b>	voltage threshold value
<i>PMU_LVDT_0</i>	voltage threshold is 2.2V
<i>PMU_LVDT_1</i>	voltage threshold is 2.3V
<i>PMU_LVDT_2</i>	voltage threshold is 2.4V
<i>PMU_LVDT_3</i>	voltage threshold is 2.5V
<i>PMU_LVDT_4</i>	voltage threshold is 2.6V
<i>PMU_LVDT_5</i>	voltage threshold is 2.7V
<i>PMU_LVDT_6</i>	voltage threshold is 2.8V
<i>PMU_LVDT_7</i>	voltage threshold is 2.9V
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select low voltage detector threshold as 2.9V */
```

```
pmu_lvd_select (PMU_LVDT_7);
```

### pmu\_ldo\_output\_select

The description of pmu\_ldo\_output\_select is shown as below:

**Table 3-356. Function pmu\_ldo\_output\_select**

<b>Function name</b>	pmu_ldo_output_select
<b>Function prototype</b>	void pmu_ldo_output_select(uint32_t ldo_output);
<b>Function descriptions</b>	internal voltage regulator (LDO) output voltage select
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ldo_output</b>	output voltage mode

<i>PMU_LDOVS_LOW</i>	output low voltage mode
<i>PMU_LDOVS_MID</i>	LDO output voltage mid mode
<i>PMU_LDOVS_HIGH</i>	LDO output voltage high mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select output low voltage mode */
```

```
pmu_ldo_output_select (PMU_LDOVS_LOW);
```

### pmu\_highdriver\_switch\_select

The description of pmu\_highdriver\_switch\_select is shown as below:

**Table 3-357. Function pmu\_highdriver\_switch\_select**

<b>Function name</b>	pmu_highdriver_switch_select
<b>Function prototype</b>	void pmu_highdriver_switch_select(uint32_t highdr_switch);
<b>Function descriptions</b>	switch high-driver mode
<b>Precondition</b>	-
<b>The called functions</b>	pmu_flag_get()
<b>Input parameter{in}</b>	
<b>highdr_switch</b>	enable or disable high-driver mode switch
<i>PMU_HIGHDR_SWITC H_NONE</i>	disable high-driver mode switch
<i>PMU_HIGHDR_SWITC H_EN</i>	enable high-driver mode switch
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable high-driver mode switch */
```

```
pmu_highdriver_switch_select (PMU_HIGHDR_SWITCH_EN);
```

### pmu\_highdriver\_mode\_enable

The description of pmu\_highdriver\_mode\_enable is shown as below:

**Table 3-358. Function pmu\_highdriver\_mode\_enable**

<b>Function name</b>	pmu_highdriver_mode_enable
<b>Function prototype</b>	void pmu_highdriver_mode_enable (void);
<b>Function descriptions</b>	enable high-driver 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 high-driver mode */
```

```
pmu_highdriver_mode_enable ();
```

### pmu\_highdriver\_mode\_disable

The description of pmu\_highdriver\_mode\_disable is shown as below:

**Table 3-359. Function pmu\_highdriver\_mode\_disable**

<b>Function name</b>	pmu_highdriver_mode_disable
<b>Function prototype</b>	void pmu_highdriver_mode_disable (void);
<b>Function descriptions</b>	disable high-driver mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

```
/* disable high-driver mode */
```

```
pmu_highdriver_mode_disable ();
```

### pmu\_lowdriver\_mode\_enable

The description of pmu\_lowdriver\_mode\_enable is shown as below:

**Table 3-360. Function pmu\_lowdriver\_mode\_enable**

<b>Function name</b>	pmu_lowdriver_mode_enable
<b>Function prototype</b>	void pmu_lowdriver_mode_enable(void);
<b>Function descriptions</b>	enable low-driver mode in deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable low-driver mode in deep-sleep mode */
```

```
pmu_lowdriver_mode_enable ();
```

### pmu\_lowdriver\_mode\_disable

The description of pmu\_lowdriver\_mode\_disable is shown as below:

Table 3-361. Function pmu\_lowdriver\_mode\_disable

Function name	pmu_lowdriver_mode_disable
Function prototype	void pmu_lowdriver_mode_disable (void);
Function descriptions	e disable low-driver mode in deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable low-driver mode in deep-sleep mode */
```

```
pmu_lowdriver_mode_disable ();
```

### pmu\_lvd\_disable

The description of pmu\_lvd\_disable is shown as below:

Table 3-362. Function pmu\_lvd\_disable

Function name	pmu_lvd_disable
Function prototype	void pmu_lvd_disable (void);
Function descriptions	disable PMU lvd
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable PMU lvd */
```

```
pmu_lvd_disable ();
```

### pmu\_lowpower\_driver\_config

The description of pmu\_lowpower\_driver\_config is shown as below:

**Table 3-363. Function pmu\_lowpower\_driver\_config**

<b>Function name</b>	pmu_lowpower_driver_config
<b>Function prototype</b>	void pmu_lowpower_driver_config(uint32_t mode);
<b>Function descriptions</b>	driver mode when use low power LDO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	driver mode
<i>PMU_NORMALDR_LO WPWR</i>	normal driver when use low power LDO
<i>PMU_LOWDR_LOWP WR</i>	low-driver mode enabled when LDEN is 11 and use low power LDO
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* normal driver when use low power LDO */
```

```
pmu_lowpower_driver_config (PMU_NORMALDR_LOWPWR);
```

### pmu\_normalpower\_driver\_config

The description of pmu\_normalpower\_driver\_config is shown as below:

Table 3-364. Function pmu\_normalpower\_driver\_config

<b>Function name</b>	pmu_normalpower_driver_config
<b>Function prototype</b>	void pmu_normalpower_driver_config (uint32_t mode);
<b>Function descriptions</b>	driver mode when use normal power LDO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	driver mode
<i>PMU_NORMALDR_NO RMALPWR</i>	normal driver when use normal power LDO
<i>PMU_LOWDR_NORM ALPWR</i>	low-driver mode enabled when LDEN is 11 and use normal power LDO
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* normal driver when use low power LDO */
```

```
pmu_normalpower_driver_config (PMU_NORMALDR_NORMALPWR);
```

### pmu\_to\_sleepmode

The description of pmu\_to\_sleepmode is shown as below:

Table 3-365. 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 at 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 3-366. Function pmu\_to\_deepsleepmode**

<b>Function name</b>	pmu_to_deepsleepmode
<b>Function prototype</b>	void pmu_to_deepsleepmode(uint32_t ldo,uint8_t deepsleepmodecmd);
<b>Function descriptions</b>	PMU work at deepsleep 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 normal work when pmu enter deepsleep mode
<i>PMU_LDO_LOWPOWER</i>	LDO work at low power mode when pmu enter deepsleep mode
<b>Input parameter{in}</b>	
<b>deepsleepmodecmd</b>	command to enter deepsleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* PMU work at deepsleep mode */
```

```
pmu_to_deepsleepmode (PMU_LDO_NORMAL, WFI_CMD);
```

### pmu\_to\_standbymode

The description of pmu\_to\_standbymode is shown as below:

**Table 3-367. Function pmu\_to\_standbymode**

Function name	pmu_to_standbymode
Function prototype	void pmu_to_standbymode(void);
Function descriptions	pmu work at standby mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PMU work at standby mode */
```

```
pmu_to_standby ();
```

### pmu\_backup\_write\_enable

The description of pmu\_backup\_write\_enable is shown as below:

**Table 3-368. Function pmu\_backup\_write\_enable**

Function name	pmu_backup_write_enable
Function prototype	void pmu_backup_write_enable (void);
Function descriptions	enable backup domain write

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

Example:

```
/* enable backup domain write */
```

```
pmu_backup_write_enable ();
```

### **pmu\_backup\_write\_disable**

The description of pmu\_backup\_write\_disable is shown as below:

**Table 3-369. Function pmu\_backup\_write\_disable**

<b>Function name</b>	pmu_backup_write_disable
<b>Function prototype</b>	void pmu_backup_write_disable (void);
<b>Function descriptions</b>	disable backup domain write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable backup domain write */
```

```
pmu_backup_write_disable ();
```

## pmu\_wakeup\_pin\_enable

The description of pmu\_wakeup\_pin\_enable is shown as below:

**Table 3-370. 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>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable wakeup pin */
pmu_wakeup_pin_enable ();
```

## pmu\_wakeup\_pin\_disable

The description of pmu\_wakeup\_pin\_disable is shown as below:

**Table 3-371. 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>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* disable wakeup pin */
```

```
pmu_wakeup_pin_disable ();
```

### pmu\_flag\_get

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

**Table 3-372. Function pmu\_flag\_get**

<b>Function name</b>	pmu_flag_get
<b>Function prototype</b>	FlagStatus pmu_flag_get(uint32_t flag);
<b>Function descriptions</b>	get flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flag
<i>PMU_FLAG_WAKEUP</i>	wakeup flag
<i>PMU_FLAG_STANDBY</i>	standby flag
<i>PMU_FLAG_LVD</i>	low voltage detector status flag
<i>PMU_FLAG_LDOVSRF</i>	LDO voltage select ready flag
<i>PMU_FLAG_HDRF</i>	high-driver ready flag
<i>PMU_FLAG_HDSRF</i>	high-driver switch ready flag
<i>PMU_FLAG_LDRF</i>	ow-driver mode ready flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get flag state */
```

```
FlagStatus status;
```

```
status = pmu_flag_get (PMU_FLAG_WAKEUP);
```

### pmu\_flag\_clear

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

**Table 3-373. Function pmu\_flag\_clear**

<b>Function name</b>	pmu_flag_clear
<b>Function prototype</b>	void pmu_flag_clear(uint32_t flag_reset);
<b>Function descriptions</b>	clear flag bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag_reset</b>	flag
PMU_FLAG_RESET_WAKEUP	reset wakeup flag
PMU_FLAG_RESET_STANDBY	reset standby flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear flag bit */
```

```
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

## 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

**Table 3-374. RCU Registers**

Registers	Descriptions
RCU_CTL	Control register
RCU_CFG0	Clock configuration register 0
RCU_INT	Clock interrupt register
RCU_APB2RST	APB2 reset register
RCU_APB1RST	APB1 reset register
RCU_AHBEN	AHB enable register
RCU_APB2EN	APB2 enable register
RCU_APB1EN	APB1 enable register
RCU_BDCTL	Backup domain control register
RCU_RSTSCK	Reset source/clock register
RCU_AHBRST	AHB reset register
RCU_CFG1	Clock configuration register 1
RCU_DSV	Deep-sleep mode voltage register
RCU_ADDCTL	Additional clock control register
RCU_ADDINT	Additional clock interrupt register
RCU_ADDAPB1RST	APB1 additional reset register
RCU_ADDAPB1EN	APB1 additional enable register

### 3.18.2. Descriptions of Peripheral functions

**Table 3-375. RCU firmware function**

Function name	Function description
rcu_deinit	deinitialize the RCU
rcu_periph_clock_enable	enable the peripherals clock
rcu_periph_clock_disable	disable the peripherals clock
rcu_periph_clock_sleep_enable	enable the peripherals clock when in sleep mode

Function name	Function description
rcu_periph_clock_sleep_disable	disable the peripherals clock when in sleep mode
rcu_periph_reset_enable	enable the peripherals reset
rcu_periph_reset_disable	disable the peripheral reset
rcu_bkp_reset_enable	enable the BKP domain reset
rcu_bkp_reset_disable	disable the BKP domain reset
rcu_system_clock_source_config	configure the system clock source
rcu_system_clock_source_get	get the system clock source
rcu_ahb_clock_config	configure the AHB clock prescaler selection
rcu_apb1_clock_config	configure the APB1 clock prescaler selection
rcu_apb2_clock_config	configure the APB2 clock prescaler selection
rcu_ckout0_config	configure the CK_OUT0 clock source
rcu_pll_config	configure the main PLL clock
rcu_pllpreselect_config	configure the PLL clock source preselection
rcu_predv0_config	configure the PREDV0 division factor
rcu_predv1_config	configure the PREDV1 division factor
rcu_pll1_config	configure the PLL1 clock
rcu_pll2_config	configure the PLL2 clock
rcu_adc_clock_config	configure the ADC prescaler factor
rcu_usb_clock_config	configure the USB prescaler factor
rcu_rtc_clock_config	configure the RTC clock source selection
rcu_i2s1_clock_config	configure the I2S1 clock source selection
rcu_i2s2_clock_config	configure the I2S2 clock source selection
rcu_ck48m_clock_config	configure the CK48M clock 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



Function name	Function description
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_hxtal_clock_monitor_enable	enable the HXTAL clock monitor
rcu_hxtal_clock_monitor_disable	disable the HXTAL clock monitor
rcu_irc8m_adjust_value_set	set the IRC8M adjust value
rcu_deepsleep_voltage_set	set the deep-sleep mode voltage value
rcu_clock_freq_get	get the system clock, bus clock frequency

## Enum rcu\_periph\_enum

Table 3-376. Enum rcu\_periph\_enum

enum name	Function description
RCU_DMA0	DMA0 clock
RCU_DMA1	DMA1 clock
RCU_CRC	CRC clock
RCU_EXMC	EXMC clock
RCU_SDIO	SDIO clock(only for HD、XD series)
RCU_USBFS	USBFS clock(only for CL series)
RCU_TIMER1	TIMER1 clock
RCU_TIMER2	TIMER2 clock
RCU_TIMER3	TIMER3 clock
RCU_TIMER4	TIMER4 clock
RCU_TIMER5	TIMER5 clock
RCU_TIMER6	TIMER6 clock
RCU_TIMER11	TIMER11 clock
RCU_TIMER12	TIMER12 clock
RCU_TIMER13	TIMER13 clock
RCU_WWDGT	WWDGT clock

enum name	Function description
RCU_SPI1	SPI1 clock
RCU_SPI2	SPI2 clock
RCU_USART1	USART1 clock
RCU_USART2	USART2 clock
RCU_UART3	UART3 clock
RCU_UART4	UART4 clock
RCU_I2C0	I2C0 clock
RCU_I2C1	I2C1 clock
RCU_CAN0	CAN0 clock
RCU_CAN1	CAN1 clock
RCU_BKPI	BKPI clock
RCU_PMU	PMU clock
RCU_DAC	DAC clock
RCU_RTC	RTC clock
RCU_CTC	CTC clock
RCU_AF	alternate function clock
RCU_GPIOA	GPIOA clock
RCU_GPIOB	GPIOB clock
RCU_GPIOC	GPIOC clock
RCU_GPIOD	GPIOD clock
RCU_GPIOE	GPIOE clock
RCU_GPIOF	GPIOF clock
RCU_GPIOG	GPIOG clock
RCU_ADC0	ADC0 clock
RCU_ADC1	ADC1 clock
RCU_TIMER0	TIMER0 clock
RCU_SPI0	SPI0 clock
RCU_TIMER7	TIMER7 clock
RCU_USART0	USART0 clock
RCU_ADC2	ADC2 clock
RCU_TIMER8	TIMER8 clock
RCU_TIMER9	TIMER9 clock
RCU_TIMER10	TIMER10 clock

### Enum rcu\_periph\_sleep\_enum

Table 3-377. Enum rcu\_periph\_sleep\_enum

enum name	Function description
RCU_SRAM_SLP	SRAM clock when sleep mode
RCU_FMC_SLP	FMC clock when sleep mode

## Enum rcu\_periph\_reset\_enum

**Table 3-378. Enum rcu\_periph\_reset\_enum**

enum name	Function description
RCU_USBFSRST	USBFS clock reset
RCU_TIMER1RST	TIMER1 clock reset
RCU_TIMER2RST	TIMER2 clock reset
RCU_TIMER3RST	TIMER3 clock reset
RCU_TIMER4RST	TIMER4 clock reset
RCU_TIMER5RST	TIMER5 clock reset
RCU_TIMER6RST	TIMER6 clock reset
RCU_TIMER11RST	TIMER11 clock reset
RCU_TIMER12RST	TIMER12 clock reset
RCU_TIMER13RST	TIMER13 clock reset
RCU_WWDGTRST	WWDGT clock reset
RCU_SPI1RST	SPI1 clock reset
RCU_SPI2RST	SPI2 clock reset
RCU_USART1RST	USART1 clock reset
RCU_USART2RST	USART2 clock reset
RCU_UART3RST	UART3 clock reset
RCU_USART4RST	UART4 clock reset
RCU_I2C0RST	I2C0 clock reset
RCU_I2C1RST	I2C1 clock reset
RCU_CAN0RST	CAN0 clock reset
RCU_CAN1RST	CAN1 clock reset
RCU_BKPIRST	BKPI clock reset
RCU_PMURST	PMU clock reset
RCU_DACRST	DAC clock reset
RCU_CTCRST	CTC clock reset
RCU_AFRST	alternate function clock reset
RCU_GPIOARST	GPIOA clock reset
RCU_GPIOBRST	GPIOB clock reset
RCU_GPIOCRST	GPIOC clock reset
RCU_GPIODRST	GPIOD clock reset
RCU_GPIOERST	GPIOE clock reset
RCU_GPIOFRST	GPIOF clock reset
RCU_GPIOGRST	GPIOG clock reset
RCU_ADC0RST	ADC0 clock reset
RCU_ADC1RST	ADC1 clock reset
RCU_TIMER0RST	TIMER0 clock reset
RCU_SPI0RST	SPI0 clock reset
RCU_TIMER7RST	TIMER7 clock reset

enum name	Function description
RCU_USART0RST	USART0 clock reset
RCU_ADC2RST	ADC2 clock reset
RCU_TIMER8RST	TIMER8 clock reset
RCU_TIMER9RST	TIMER9 clock reset
RCU_TIMER10RST	TIMER10 clock reset

### Enum rcu\_flag\_enum

**Table 3-379. Enum rcu\_flag\_enum**

enum name	Function description
RCU_FLAG_IRC8MSTB	IRC8M stabilization flags
RCU_FLAG_HXTALSTB	HXTAL stabilization flags
RCU_FLAG_PLLSTB	PLL stabilization flags
RCU_FLAG_PLL1STB	PLL1 stabilization flags
RCU_FLAG_PLL2STB	PLL2 stabilization flags
RCU_FLAG_LXTALSTB	LXTAL stabilization flags
RCU_FLAG_IRC40KSTB	IRC40K stabilization flags
RCU_FLAG_IRC48MSTB	IRC48M stabilization flags
RCU_FLAG_EPRST	external PIN reset flags
RCU_FLAG_PORRST	power reset flags
RCU_FLAG_SWRST	software reset flags
RCU_FLAG_FWDGTRST	FWDGT reset flags
RCU_FLAG_WWDGTRST	WWDGT reset flags
RCU_FLAG_LPRST	low-power reset flags

### Enum rcu\_int\_flag\_enum

**Table 3-380. Enum rcu\_int\_flag\_enum**

enum name	Function description
RCU_INT_FLAG_IRC40KSTB	IRC40K stabilization interrupt flag
RCU_INT_FLAG_LXTALSTB	LXTAL stabilization interrupt flag
RCU_INT_FLAG_IRC8MSTB	IRC8M stabilization interrupt flag
RCU_INT_FLAG_HXTALSTB	HXTAL stabilization interrupt flag
RCU_INT_FLAG_PLLSTB	PLL stabilization interrupt flag
RCU_INT_FLAG_PLL1STB	PLL1 stabilization interrupt flag
RCU_INT_FLAG_PLL2STB	PLL2 stabilization interrupt flag
RCU_INT_FLAG_CKM	HXTAL clock stuck interrupt flag
RCU_INT_FLAG_IRC48MSTB	IRC48M stabilization interrupt flag

## Enum rcu\_int\_flag\_clear\_enum

**Table 3-381. Enum rcu\_int\_flag\_clear\_enum**

enum name	Function description
RCU_INT_FLAG_IRC40KSTB_CLR	IRC40K stabilization interrupt flags clear
RCU_INT_FLAG_LXTALSTB_CLR	LXTAL stabilization interrupt flags clear
RCU_INT_FLAG_IRC8MSTB_CLR	IRC8M stabilization interrupt flags clear
RCU_INT_FLAG_HXTALSTB_CLR	HXTAL stabilization interrupt flags clear
RCU_INT_FLAG_PLLSTB_CLR	PLL stabilization interrupt flags clear
RCU_INT_FLAG_PLL1STB_CLR	PLL1 stabilization interrupt flags clear
RCU_INT_FLAG_PLL2STB_CLR	PLL2 stabilization interrupt flags clear
RCU_INT_FLAG_CKM_CLR	CKM interrupt flags clear
RCU_INT_FLAG_IRC48MSTB_CLR	internal 48 MHz RC oscillator stabilization interrupt clear

## Enum rcu\_int\_enum

**Table 3-382. Enum rcu\_int\_enum**

enum name	Function description
RCU_INT_IRC40KSTB	IRC40K stabilization interrupt
RCU_INT_LXTALSTB	LXTAL stabilization interrupt
RCU_INT_IRC8MSTB	IRC8M stabilization interrupt
RCU_INT_HXTALSTB	HXTAL stabilization interrupt
RCU_INT_PLLSTB	PLL stabilization interrupt
RCU_INT_PLL1STB	PLL1 stabilization interrupt(only for CL series)
RCU_INT_PLL2STB	PLL2 stabilization interrupt(only for CL series)
RCU_INT_IRC48MSTB	internal 48 MHz RC oscillator stabilization interrupt

## Enum rcu\_osci\_type\_enum

**Table 3-383. Enum rcu\_osci\_type\_enum**

enum name	Function description
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL
RCU_IRC8M	IRC8M
RCU_IRC48M	IRC48M
RCU_IRC40K	IRC40K
RCU_PLL_CK	PLL
RCU_PLL1_CK	PLL1(only for CL series)
RCU_PLL2_CK	PLL2(only for CL series)

## Enum rcu\_clock\_freq\_enum

**Table 3-384. 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

## rcu\_deinit

The description of rcu\_deinit is shown as below:

**Table 3-385. Function rcu\_deinit**

<b>Function name</b>	rcu_deinit
<b>Function prototype</b>	void rcu_deinit(void);
<b>Function descriptions</b>	deinitialize the RCU
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset RCU */
rcu_deinit();
```

## rcu\_periph\_clock\_enable

The description of rcu\_periph\_clock\_enable is shown as below:

**Table 3-386. Function rcu\_periph\_clock\_enable**

<b>Function name</b>	rcu_periph_clock_enable
<b>Function prototype</b>	void rcu_periph_clock_enable(rcu_periph_enum periph);

<b>Function descriptions</b>	enable the peripherals clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-376. Enum rcu_periph_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-387. Function rcu\_periph\_clock\_disable**

<b>Function name</b>	rcu_periph_clock_disable
<b>Function prototype</b>	void rcu_periph_clock_disable(rcu_periph_enum periph);
<b>Function descriptions</b>	disable the peripherals clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-376. Enum rcu_periph_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the USART0 clock */
```

```
rcu_periph_clock_disable(RCU_USART0);
```

## rcu\_periph\_clock\_sleep\_enable

The description of rcu\_periph\_clock\_sleep\_enable is shown as below:

**Table 3-388. Function rcu\_periph\_clock\_sleep\_enable**

<b>Function name</b>	rcu_periph_clock_sleep_enable
<b>Function prototype</b>	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
<b>Function descriptions</b>	enable the peripherals clock when in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-377. Enum rcu_periph_sleep_enum</a>
<i>RCU_FMC_SLP</i>	FMC clock
<i>RCU_SRAM_SLP</i>	SRAM clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the FMC clock when in sleep mode */
```

```
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

## rcu\_periph\_clock\_sleep\_disable

The description of rcu\_periph\_clock\_sleep\_disable is shown as below:

**Table 3-389. Function rcu\_periph\_clock\_sleep\_disable**

<b>Function name</b>	rcu_periph_clock_sleep_disable
<b>Function prototype</b>	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);
<b>Function descriptions</b>	disable the peripherals clock when in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-



Input parameter{in}	
<b>periph</b>	RCU peripherals, refer to <a href="#">Table 3-377. Enum rcu_periph_sleep_enum</a>
<i>RCU_FMC_SLP</i>	FMC clock
<i>RCU_SRAM_SLP</i>	SRAM clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the FMC clock when in sleep mode */
```

```
rcu_periph_clock_sleep_disable(RCU_FMC_SLP);
```

### rcu\_periph\_reset\_enable

The description of rcu\_periph\_reset\_enable is shown as below:

**Table 3-390. Function rcu\_periph\_reset\_enable**

<b>Function name</b>	rcu_periph_reset_enable
<b>Function prototype</b>	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
<b>Function descriptions</b>	enable the peripherals reset
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>periph_reset</b>	RCU peripherals reset, refer to <a href="#">Table 3-378. Enum rcu_periph_reset_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 reset */
```

```
rcu_periph_reset_enable(RCU_SPI0RST);
```

## rcu\_periph\_reset\_disable

The description of rcu\_periph\_reset\_disable is shown as below:

**Table 3-391. Function rcu\_periph\_reset\_disable**

<b>Function name</b>	rcu_periph_reset_disable
<b>Function prototype</b>	void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);
<b>Function descriptions</b>	disable the peripheral reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph_reset</b>	RCU peripherals reset, refer to <a href="#">Table 3-378. Enum rcu_periph_reset_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 reset */
```

```
rcu_periph_reset_disable(RCU_SPI0RST);
```

## rcu\_bkp\_reset\_enable

The description of rcu\_bkp\_reset\_enable is shown as below:

**Table 3-392. Function rcu\_bkp\_reset\_enable**

<b>Function name</b>	rcu_bkp_reset_enable
<b>Function prototype</b>	void rcu_bkp_reset_enable(void);
<b>Function descriptions</b>	enable the BKP domain reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

-	-
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 3-393. Function rcu\_bkp\_reset\_disable**

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

Example:

```
/* disable the BKP domain reset */
```

```
rcu_bkp_reset_disable();
```

### rcu\_system\_clock\_source\_config

The description of rcu\_system\_clock\_source\_config is shown as below:

Table 3-394. 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_IRC8M</i>	select CK_IRC8M as the CK_SYS source
<i>RCU_CKSYSSRC_HXTAL</i>	select CK_HXTAL as the CK_SYS source
<i>RCU_CKSYSSRC_PLL</i>	select CK_PLL as the CK_SYS source
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CK_HXTAL as the CK_SYS source */
rcu_system_clock_source_config(RCU_CKSYSSRC_HXTAL);
```

### rcu\_system\_clock\_source\_get

The description of rcu\_system\_clock\_source\_get is shown as below:

Table 3-395. Function rcu\_system\_clock\_source\_get

<b>Function name</b>	rcu_system_clock_source_get
<b>Function prototype</b>	uint32_t rcu_system_clock_source_get(void);
<b>Function descriptions</b>	get the system clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

-	-
Output parameter{out}	
-	-
Return value	
uint32_t	RCU_SCSS_IRC8M / RCU_SCSS_HXTAL / RCU_SCSS_PLL

Example:

```
uint32_t temp_cksys_status;
```

```
/* get the CK_SYS source */
```

```
temp_cksys_status = rcu_system_clock_source_get();
```

### rcu\_ahb\_clock\_config

The description of rcu\_ahb\_clock\_config is shown as below:

**Table 3-396. Function rcu\_ahb\_clock\_config**

Function name	rcu_ahb_clock_config
Function prototype	void rcu_ahb_clock_config(uint32_t ck_ahb);
Function descriptions	configure the AHB clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
ck_ahb	AHB clock prescaler selection
RCU_AHB_CKSYS_DIVx	select CK_SYS / x, (x = 1, 2, 4, 8, 16, 64, 128, 256, 512)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CK_SYS/128 */
```

```
rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

## rcu\_apb1\_clock\_config

The description of rcu\_apb1\_clock\_config is shown as below:

**Table 3-397. 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_D</i> <i>IV1</i>	select CK_AHB as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV2</i>	select CK_AHB / 2 as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV4</i>	select CK_AHB / 4 as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV8</i>	select CK_AHB / 8 as CK_APB1
<i>RCU_APB1_CKAHB_D</i> <i>IV16</i>	select CK_AHB / 16 as CK_APB1
<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 3-398. Function rcu\_apb2\_clock\_config

<b>Function name</b>	rcu_apb2_clock_config
<b>Function prototype</b>	void rcu_apb2_clock_config(uint32_t ck_apb2);
<b>Function descriptions</b>	configure the APB2 clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_apb2</b>	APB2 clock prescaler selection
<i>RCU_APB2_CKAHB_DIV1</i>	select CK_AHB as CK_APB2
<i>RCU_APB2_CKAHB_DIV2</i>	select CK_AHB / 2 as CK_APB2
<i>RCU_APB2_CKAHB_DIV4</i>	select CK_AHB / 4 as CK_APB2
<i>RCU_APB2_CKAHB_DIV8</i>	select CK_AHB / 8 as CK_APB2
<i>RCU_APB2_CKAHB_DIV16</i>	select CK_AHB / 16 as CK_APB2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure CK_AHB/8 as CK_APB2 */
rcu_apb2_clock_config(RCU_APB2_CKAHB_DIV8);
```

### rcu\_ckout0\_config

The description of rcu\_ckout0\_config is shown as below:

Table 3-399. Function rcu\_ckout0\_config

<b>Function name</b>	rcu_ckout0_config
<b>Function prototype</b>	void rcu_ckout0_config(uint32_t ckout0_src);

<b>Function descriptions</b>	configure the CK_OUT0 clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ckout0_src</b>	CK_OUT0 clock source selection
<i>RCU_CKOUT0SRC_NONE</i>	no clock selected
<i>RCU_CKOUT0SRC_CKSYS</i>	select system clock CK_SYS
<i>RCU_CKOUT0SRC_IRC8M</i>	select high speed 8M internal oscillator clock
<i>RCU_CKOUT0SRC_HXTAL</i>	select HXTAL
<i>RCU_CKOUT0SRC_CKPLL_DIV2</i>	select (CK_PLL / 2) clock
<i>RCU_CKOUT0SRC_CKPLL1</i>	select CK_PLL1 clock
<i>RCU_CKOUT0SRC_CKPLL2_DIV2</i>	select (CK_PLL2 / 2) clock
<i>RCU_CKOUT0SRC_CKPLL2</i>	select CK_PLL2 clock
<i>RCU_CKOUT0SRC_EXT1</i>	select EXT1 clock
<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\_pll\_config

The description of rcu\_pll\_config is shown as below:

**Table 3-400. Function rcu\_pll\_config**

<b>Function name</b>	rcu_pll_config
<b>Function prototype</b>	void rcu_pll_config(uint32_t pll_src, uint32_t pll_mul);
<b>Function descriptions</b>	configure the main PLL clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_src</b>	PLL clock source selection
<i>RCU_PLLSRC_IRC8M_DIV2</i>	IRC8M/2 clock is selected as source clock of PLL
<i>RCU_PLLSRC_HXTAL_IRC48M</i>	HXTAL or IRC48M is selected as source clock of PLL
<b>Input parameter{in}</b>	
<b>pll_mul</b>	PLL clock multiplication factor
<i>RCU_PLL_MULx</i>	PLL clock * x (x = 2..14, 6.5, 16..63)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL */
```

```
rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PLL_MUL10);
```

## rcu\_pllpresel\_config

The description of rcu\_pllpresel\_config is shown as below:

**Table 3-401. Function rcu\_pllpresel\_config**

<b>Function name</b>	rcu_pllpresel_config
<b>Function prototype</b>	void rcu_pllpresel_config(uint32_t pll_presel);

<b>Function descriptions</b>	configure the PLL clock source preselection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_presel</b>	PLL clock source preselection
<i>RCU_PLLPRESRC_HXTAL</i>	HXTAL selected as PREDV0 source clock
<i>RCU_PLLPRESRC_IRC48M</i>	CK_PLL selected as PREDV0 input source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL clock source preselection */
```

```
rcu_pllpresel_config (RCU_PLLPRESRC_HXTAL);
```

### rcu\_predv0\_config

The description of rcu\_predv0\_config is shown as below:

**Table 3-402. Function rcu\_predv0\_config**

<b>Function name</b>	rcu_predv0_config
<b>Function prototype</b>	void rcu_predv0_config(uint32_t predv0_source, uint32_t predv0_div);
<b>Function descriptions</b>	configure the PREDV0 division factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>predv0_source</b>	PREDV0 input clock source selection
<i>RCU_PREDV0SRC_HXTAL_IRC48M</i>	select HXTAL or IRC48M as PREDV0 input source clock
<i>RCU_PREDV0SRC_CKPLL1</i>	select CK_PLL1 as PREDV0 input source clock

<i>KPLL1</i>	
<b>Input parameter{in}</b>	
<b>predv0_div</b>	PREDV0 division factor
<i>RCU_PREDV0_DIVx</i>	PREDV0 input source clock is divided x (x = 1..16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PREDV0 division factor */
```

```
rcu_predv0_config(RCU_PREDV0SRC_HXTAL_IRC48M, RCU_PREDV0_DIV4);
```

### rcu\_predv1\_config

The description of rcu\_predv1\_config is shown as below:

**Table 3-403. Function rcu\_predv1\_config**

<b>Function name</b>	rcu_predv1_config
<b>Function prototype</b>	void rcu_predv1_config(uint32_t predv1_div);
<b>Function descriptions</b>	configure the PREDV1 division factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>predv1_div</b>	PREDV1 division factor
<i>RCU_PREDV1_DIVx</i>	PREDV1 input source clock is divided x (x = 1..16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PREDV1 division factor */
```

```
rcu_predv1_config(RCU_PREDV1_DIV8);
```

## rcu\_pll1\_config

The description of rcu\_pll1\_config is shown as below:

**Table 3-404. Function rcu\_pll1\_config**

<b>Function name</b>	rcu_pll1_config
<b>Function prototype</b>	void rcu_pll1_config(uint32_t pll_mul);
<b>Function descriptions</b>	configure the PLL1 clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll_mul</b>	PLL clock multiplication factor
<i>RCU_PLL1_MULx</i>	PLL1 clock * x, (x = 8..14,16,20)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL1 clock */
```

```
rcu_pll1_config(RCU_PLL1_MUL8);
```

## rcu\_pll2\_config

The description of rcu\_pll2\_config is shown as below:

**Table 3-405. Function rcu\_pll2\_config**

<b>Function name</b>	rcu_pll2_config
<b>Function prototype</b>	void rcu_pll2_config(uint32_t pll_mul)
<b>Function descriptions</b>	configure the PLL2 clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>pll_mul</b>	PLL clock multiplication factor
<i>RCU_PLL2_MULx</i>	PLL2 clock * x, (x = 8..14,16,20,18..32,40)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL2 clock */
```

```
rcu_pll2_config(RCU_PLL2_MUL8);
```

### rcu\_adc\_clock\_config

The description of rcu\_adc\_clock\_config is shown as below:

**Table 3-406. Function rcu\_adc\_clock\_config**

<b>Function name</b>	rcu_adc_clock_config
<b>Function prototype</b>	void rcu_adc_clock_config(uint32_t adc_psc);
<b>Function descriptions</b>	configure the ADC prescaler factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_psc</b>	ADC prescaler factor
<i>RCU_CKADC_CKAPB2_DIV2</i>	CK_ADC = CK_APB2 / 2
<i>RCU_CKADC_CKAPB2_DIV4</i>	CK_ADC = CK_APB2 / 4
<i>RCU_CKADC_CKAPB2_DIV6</i>	CK_ADC = CK_APB2 / 6
<i>RCU_CKADC_CKAPB2_DIV8</i>	CK_ADC = CK_APB2 / 8
<i>RCU_CKADC_CKAPB2_DIV12</i>	CK_ADC = CK_APB2 / 12
<i>RCU_CKADC_CKAPB</i>	CK_ADC = CK_APB2 / 16

<i>2_DIV16</i>	
<i>RCU_CKADC_CKAHB _DIV5</i>	$CK\_ADC = CK\_AHB / 5$
<i>RCU_CKADC_CKAHB _DIV6</i>	$CK\_ADC = CK\_AHB / 6$
<i>RCU_CKADC_CKAHB _DIV10</i>	$CK\_ADC = CK\_AHB / 10$
<i>RCU_CKADC_CKAHB _DIV20</i>	$CK\_ADC = CK\_AHB / 20$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the ADC prescaler factor */
```

```
rcu_adc_clock_config(RCU_CKADC_CKAPB2_DIV8);
```

### rcu\_usb\_clock\_config

The description of rcu\_usb\_clock\_config is shown as below:

**Table 3-407. Function rcu\_usb\_clock\_config**

<b>Function name</b>	rcu_usb_clock_config
<b>Function prototype</b>	void rcu_usb_clock_config(uint32_t usb_psc);
<b>Function descriptions</b>	configure the USB prescaler factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usb_psc</b>	USB prescaler factor
<i>RCU_CKUSB_CKPLL_ DIV1_5</i>	$CK\_USBFS = CK\_PLL / 1.5$
<i>RCU_CKUSB_CKPLL_ DIV1</i>	$CK\_USBFS = CK\_PLL / 1$

<i>RCU_CKUSB_CKPLL_DIV2_5</i>	$CK\_USBFS = CK\_PLL / 2.5$
<i>RCU_CKUSB_CKPLL_DIV2</i>	$CK\_USBFS = CK\_PLL / 2$
<i>RCU_CKUSB_CKPLL_DIV3</i>	$CK\_USBFS = CK\_PLL / 3$
<i>RCU_CKUSB_CKPLL_DIV3_5</i>	$CK\_USBFS = CK\_PLL / 3.5$
<i>RCU_CKUSB_CKPLL_DIV4</i>	$CK\_USBFS = CK\_PLL / 4$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the USB prescaler factor */
```

```
rcu_usb_clock_config(RCU_CKUSB_CKPLL_DIV2_5);
```

### rcu\_rtc\_clock\_config

The description of rcu\_rtc\_clock\_config is shown as below:

**Table 3-408. 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_IRC40</i>	select CK_IRC40K as RTC source clock

<i>K</i>	
<i>RCU_RTCSRC_HXTAL</i> <i>_DIV_128</i>	select CK_HXTAL/128 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_IRC40K);
```

### rcu\_i2s1\_clock\_config

The description of rcu\_i2s1\_clock\_config is shown as below:

**Table 3-409. Function rcu\_i2s1\_clock\_config**

<b>Function name</b>	rcu_i2s1_clock_config
<b>Function prototype</b>	void rcu_i2s1_clock_config(uint32_t i2s_clock_source);
<b>Function descriptions</b>	configure the I2S1 clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2s_clock_source</b>	I2S clock source selection
<i>RCU_I2S1SRC_CKSYS</i>	select system clock as I2S1 source clock
<i>RCU_I2S1SRC_CKPLL2_MUL2</i>	select CK_PLL2 * 2 as I2S1 source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:



```
/* configure the I2S1 clock source selection */
```

```
rcu_i2s1_clock_config(RCU_I2S1SRC_CKPLL2_MUL2);
```

### rcu\_i2s2\_clock\_config

The description of rcu\_i2s2\_clock\_config is shown as below:

**Table 3-410. Function rcu\_i2s2\_clock\_config**

<b>Function name</b>	rcu_i2s2_clock_config
<b>Function prototype</b>	void rcu_i2s2_clock_config(uint32_t i2s_clock_source);
<b>Function descriptions</b>	configure the I2S2 clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2s_clock_source</b>	I2S clock source selection
<i>RCU_I2S2SRC_CKSYS</i>	select system clock as I2S2 source clock
<i>RCU_I2S2SRC_CKPLL2_MUL2</i>	select CK_PLL2 * 2 as I2S2 source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the I2S2 clock source selection */
```

```
rcu_i2s2_clock_config(RCU_I2S2SRC_CKPLL2_MUL2);
```

### rcu\_ck48m\_clock\_config

The description of rcu\_ck48m\_clock\_config is shown as below:

**Table 3-411. Function rcu\_ck48m\_clock\_config**

<b>Function name</b>	rcu_ck48m_clock_config
<b>Function prototype</b>	void rcu_ck48m_clock_config(uint32_t ck48m_clock_source);
<b>Function descriptions</b>	configure the CK48M clock source selection

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck48m_clock_source</b>	CK48M clock source selection
<i>RCU_CK48MSRC_CKPLL</i>	CK_PLL selected as CK48M source clock
<i>RCU_CK48MSRC_IRC48M</i>	CK_IRC48M selected as CK48M source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CK48M clock source selection */
```

```
rcu_ck48m_clock_config(RCU_CK48MSRC_CKPLL);
```

### rcu\_flag\_get

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

**Table 3-412. 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 3-379. 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 3-413. 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 3-414. 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 3-380. Enum rcu_int_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 3-415. 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 3-381. 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 3-416. 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 3-382. 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 3-417. 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 3-382. 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 3-418. 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_LXTAL_LOWDRI</i>	lower driving capability
<i>RCU_LXTAL_MED_LOWDRI</i>	medium low driving capability
<i>RCU_LXTAL_MED_HIGHDRI</i>	medium high driving capability
<i>RCU_LXTAL_HIGHDRI</i>	higher driving capability
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the LXTAL drive capability */
```

```
rcu_lxtal_drive_capability_config (RCU_LXTAL_LOWDRI);
```

## rcu\_osci\_stab\_wait

The description of rcu\_osci\_stab\_wait is shown as below:

**Table 3-419. 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 3-383. 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 3-420. 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>	-

Input parameter{in}	
osci	oscillator types, refer to <a href="#">Table 3-383. Enum rcu_osci_type_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn on the high speed crystal oscillator */
```

```
rcu_osci_on(RCU_HXTAL);
```

### rcu\_osci\_off

The description of rcu\_osci\_off is shown as below:

**Table 3-421. Function rcu\_osci\_off**

Function name	rcu_osci_off
Function prototype	void rcu_osci_off(rcu_osci_type_enum osci);
Function descriptions	turn off the oscillator
Precondition	-
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to <a href="#">Table 3-383. Enum rcu_osci_type_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn off the high speed crystal oscillator */
```

```
rcu_osci_off(RCU_HXTAL);
```

### rcu\_osci\_bypass\_mode\_enable

The description of rcu\_osci\_bypass\_mode\_enable is shown as below:



Table 3-422. Function rcu\_osci\_bypass\_mode\_enable

Function name	rcu_osci_bypass_mode_enable
Function prototype	void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);
Function descriptions	enable the oscillator bypass mode
Precondition	HXTALEN or LXTALEN must be reset before it
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to <a href="#">Table 3-383. Enum rcu_osci_type_enum</a>
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

### rcu\_osci\_bypass\_mode\_disable

The description of rcu\_osci\_bypass\_mode\_disable is shown as below:

Table 3-423. Function rcu\_osci\_bypass\_mode\_disable

Function name	rcu_osci_bypass_mode_disable
Function prototype	void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci);
Function descriptions	disable the oscillator bypass mode
Precondition	HXTALEN or LXTALEN must be reset before it
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to <a href="#">Table 3-383. Enum rcu_osci_type_enum</a>
RCU_HXTAL	high speed crystal oscillator(HXTAL)

<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the high speed crystal oscillator bypass mode */
```

```
rcu_osc_bypass_mode_disable(RCU_HXTAL);
```

### **rcu\_hxtal\_clock\_monitor\_enable**

The description of rcu\_hxtal\_clock\_monitor\_enable is shown as below:

**Table 3-424. Function rcu\_hxtal\_clock\_monitor\_enable**

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

Example:

```
/* enable the HXTAL clock monitor */
```

```
rcu_hxtal_clock_monitor_enable();
```

### **rcu\_hxtal\_clock\_monitor\_disable**

The description of rcu\_hxtal\_clock\_monitor\_disable is shown as below:

Table 3-425. Function rcu\_hxtal\_clock\_monitor\_disable

Function name	rcu_hxtal_clock_monitor_disable
Function prototype	void rcu_hxtal_clock_monitor_disable(void);
Function descriptions	disable the HXTAL clock monitor
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the HXTAL clock monitor */
rcu_hxtal_clock_monitor_disable();
```

### rcu\_irc8m\_adjust\_value\_set

The description of rcu\_irc8m\_adjust\_value\_set is shown as below:

Table 3-426. Function rcu\_irc8m\_adjust\_value\_set

Function name	rcu_irc8m_adjust_value_set
Function prototype	void rcu_irc8m_adjust_value_set(uint32_t irc8m_adjval);
Function descriptions	set the IRC8M adjust value
Precondition	-
The called functions	-
Input parameter{in}	
irc8m_adjval	IRC8M adjust value, must be between 0 and 0x1F
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* set the IRC8M adjust value */
```

```
rcu_irc8m_adjust_value_set(0x10);
```

### rcu\_deepsleep\_voltage\_set

The description of rcu\_deepsleep\_voltage\_set is shown as below:

**Table 3-427. Function rcu\_deepsleep\_voltage\_set**

<b>Function name</b>	rcu_deepsleep_voltage_set
<b>Function prototype</b>	void rcu_deepsleep_voltage_set(uint32_t dsvol);
<b>Function descriptions</b>	set the deep-sleep mode voltage value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dsvol</b>	deep sleep mode voltage
<i>RCU_DEEPSLEEP_V_0</i>	the core voltage is default value in deep-sleep mode
<i>RCU_DEEPSLEEP_V_1</i>	the core voltage is (default value-0.1)V in deep-sleep mode(customers are not recommended to use it)
<i>RCU_DEEPSLEEP_V_2</i>	the core voltage is (default value-0.2)V in deep-sleep mode(customers are not recommended to use it)
<i>RCU_DEEPSLEEP_V_3</i>	the core voltage is (default value-0.3)V in deep-sleep mode(customers are not recommended to use it)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the deep-sleep mode voltage */
```

```
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1);
```

## rcu\_clock\_freq\_get

The description of rcu\_clock\_freq\_get is shown as below:

**Table 3-428. Function rcu\_clock\_freq\_get**

<b>Function name</b>	rcu_clock_freq_get
<b>Function prototype</b>	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
<b>Function descriptions</b>	get the system clock, bus clock frequency
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>clock</b>	the clock frequency which to get, refer to <a href="#">Table 3-384. Enum rcu_clock_freq_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ck_freq</b>	clock frequency of system, AHB, APB1, APB2

Example:

```
uint32_t temp_freq;

/* get the system clock frequency */

temp_freq = rcu_clock_freq_get(CK_SYS);
```

## 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, an alarm, 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 3-429. RTC Registers**

Registers	Descriptions
RTC_INTEN	Interrupt enable register
RTC_CTL	Control register
RTC_PSCH	Prescaler high register
RTC_PSCL	Prescaler low register
RTC_DIVH	Divider high register
RTC_DIVL	Divider low register
RTC_CNTH	counter high register
RTC_CNTL	counter low register
RTC_ALRMH	Alarm high register
RTC_ALRML	Alarm low register

### 3.19.2. Descriptions of Peripheral functions

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

**Table 3-430. RTC firmware function**

Function name	Function description
rtc_interrupt_enable	enable RTC interrupt
rtc_interrupt_disable	disable RTC interrupt
rtc_configuration_mode_enter	enter RTC configuration mode
rtc_configuration_mode_exit	exit RTC configuration mode
rtc_lwoff_wait	wait RTC last write operation finished flag set
rtc_register_sync_wait	wait RTC registers synchronized flag set
rtc_counter_get	get RTC counter value
rtc_counter_set	set RTC counter value
rtc_prescaler_set	set RTC prescaler value
rtc_alarm_config	set RTC alarm value
rtc_divider_get	get RTC divider value
rtc_flag_get	get RTC flag status

Function name	Function description
rtc_flag_clear	clear RTC flag status

### rtc\_interrupt\_enable

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

**Table 3-431. Function rtc\_interrupt\_enable**

Function name	rtc_interrupt_enable
Function prototype	void rtc_interrupt_enable(uint32_t interrupt);
Function descriptions	enable RTC interrupt
Precondition	before using this function, you must call rtc_lwoff_wait ( ) function (wait until LWOFF flag is set).
The called functions	-
Input parameter{in}	
interrupt	specify which RTC interrupt to enable
RTC_INT_SECOND	second interrupt
RTC_INT_ALARM	alarm interrupt
RTC_INT_OVERFLOW	overflow interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* wait until last write operation on RTC registers has finished */
rtc_lwoff_wait( );

/* enable the RTC second interrupt */
rtc_interrupt_enable(RTC_INT_SECOND);

```

### rtc\_interrupt\_disable

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

Table 3-432. Function `rtc_interrupt_disable`

<b>Function name</b>	<code>rtc_interrupt_disable</code>
<b>Function prototype</b>	<code>void rtc_interrupt_disable(uint32_t interrupt);</code>
<b>Function descriptions</b>	disable RTC interrupt
<b>Precondition</b>	before using this function, you must call <code>rtc_lwoff_wait ( )</code> function (wait until LWOFF flag is set).
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which RTC interrupt to disable
<code>RTC_INT_SECOND</code>	second interrupt
<code>RTC_INT_ALARM</code>	alarm interrupt
<code>RTC_INT_OVERFLOW</code>	overflow interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait( );
```

```
/* disable the RTC second interrupt */
```

```
rtc_interrupt_disable(RTC_INT_SECOND);
```

### **`rtc_configuration_mode_enter`**

The description of `rtc_configuration_mode_enter` is shown as below:

Table 3-433. Function `rtc_configuration_mode_enter`

<b>Function name</b>	<code>rtc_configuration_mode_enter</code>
<b>Function prototype</b>	<code>void rtc_configuration_mode_enter(void);</code>
<b>Function descriptions</b>	enter RTC configuration mode
<b>Precondition</b>	-
<b>The called functions</b>	-



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

Example:

```
/* enter RTC configuration mode */
```

```
rtc_configuration_mode_enter( );
```

### rtc\_configuration\_mode\_exit

The description of rtc\_configuration\_mode\_exit is shown as below:

**Table 3-434. Function rtc\_configuration\_mode\_exit**

Function name	rtc_configuration_mode_exit
Function prototype	void rtc_configuration_mode_exit(void);
Function descriptions	exit RTC configuration mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* exit RTC configuration mode */
```

```
rtc_configuration_mode_exit ( );
```

### rtc\_lwoff\_wait

The description of rtc\_lwoff\_wait is shown as below:

Table 3-435. Function rtc\_lwoff\_wait

Function name	rtc_lwoff_wait
Function prototype	void rtc_lwoff_wait(void);
Function descriptions	wait RTC last write operation finished flag set
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* wait until last write operation on RTC registers has finished */
rtc_lwoff_wait( );

/* enable the RTC second interrupt */
rtc_interrupt_enable(RTC_INT_SECOND);

```

### rtc\_register\_sync\_wait

The description of rtc\_register\_sync\_wait is shown as below:

Table 3-436. Function rtc\_register\_sync\_wait

Function name	rtc_register_sync_wait
Function prototype	void rtc_register_sync_wait(void);
Function descriptions	wait RTC registers synchronized flag set
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* wait for RTC registers synchronization */
```

```
rtc_register_sync_wait( );
```

### rtc\_counter\_get

The description of rtc\_counter\_get is shown as below:

**Table 3-437. Function rtc\_counter\_get**

Function name	rtc_counter_get
Function prototype	uint32_t rtc_counter_get(void);
Function descriptions	get RTC counter value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the value of RTC counter

Example:

```
/* get the counter value */
```

```
uint32_t rtc_counter_value;
```

```
rtc_counter_value = rtc_counter_get ( );
```

### rtc\_counter\_set

The description of rtc\_counter\_set is shown as below:

**Table 3-438. Function rtc\_counter\_set**

Function name	rtc_counter_set
---------------	-----------------

<b>Function prototype</b>	void rtc_counter_set(uint32_t cnt);
<b>Function descriptions</b>	set RTC counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cnt</b>	RTC counter value (0-0xFFFF FFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait( );
```

```
/* set counter value to 0xFFFF */
```

```
rtc_counter_set (0xFFFF);
```

### rtc\_prescaler\_set

The description of rtc\_prescaler\_set is shown as below:

**Table 3-439. Function rtc\_prescaler\_set**

<b>Function name</b>	rtc_interrupt_rtc_prescaler_set
<b>Function prototype</b>	void rtc_prescaler_set(uint32_t psc);
<b>Function descriptions</b>	set RTC prescaler value
<b>Precondition</b>	before using this function, you must call rtc_lwoff_wait ( ) function (wait until LWOFF flag is set).
<b>The called functions</b>	rtc_configuration_mode_enter / rtc_configuration_mode_exit
<b>Input parameter{in}</b>	
<b>psc</b>	RTC prescaler value (0-0x000F FFFF)
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait( );
```

```
/* set RTC prescaler value to 0x7FFFF */
```

```
rtc_prescaler_set (0x7FFFF);
```

### rtc\_alarm\_config

The description of rtc\_alarm\_config is shown as below:

**Table 3-440. Function rtc\_alarm\_config**

<b>Function name</b>	rtc_alarm_config
<b>Function prototype</b>	void rtc_alarm_config(uint32_t alarm);
<b>Function descriptions</b>	set RTC alarm value
<b>Precondition</b>	before using this function, you must call rtc_lwoff_wait ( ) function (wait until LWOFF flag is set). -
<b>The called functions</b>	rtc_configuration_mode_enter / rtc_configuration_mode_exit
<b>Input parameter{in}</b>	
<b>alarm</b>	RTC alarm value (0-0xFFFF FFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait( );
```

```
/* set alarm value to 0xFFFF */
```

```
rtc_alarm_config (0xFFFF);
```

## rtc\_divider\_get

The description of rtc\_divider\_get is shown as below:

**Table 3-441. Function rtc\_divider\_get**

<b>Function name</b>	rtc_divider_get
<b>Function prototype</b>	uint32_t rtc_divider_get(void);
<b>Function descriptions</b>	get RTC divider value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	the value of RTC divider

Example:

```
/* get the current RTC divider value */
uint32_t rtc_divider_value;
rtc_divider_value = rtc_divider_get ( );
```

## rtc\_flag\_get

The description of rtc\_flag\_getrtc\_interrupt\_enable is shown as below:

**Table 3-442. 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>	get RTC flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
flag	specify which RTC flag status to get

<i>RTC_FLAG_SECOND</i>	second interrupt flag
<i>RTC_FLAG_ALARM</i>	alarm interrupt flag
<i>RTC_FLAG_OVERFLOW</i> <i>W</i>	overflow interrupt flag
<i>RTC_FLAG_RSYN</i>	registers synchronized flag
<i>RTC_FLAG_LWOF</i>	last write operation finished flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the RTC overflow interrupt status */
```

```
FlagStatus alarm_status;
```

```
alarm_status = rtc_flag_get (RTC_FLAG_ALARM);
```

### rtc\_flag\_clear

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

**Table 3-443. 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 RTC flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which RTC flag status to clear
<i>RTC_FLAG_SECOND</i>	second interrupt flag
<i>RTC_FLAG_ALARM</i>	alarm interrupt flag
<i>RTC_FLAG_OVERFLOW</i> <i>W</i>	overflow interrupt flag
<i>RTC_FLAG_RSYN</i>	registers synchronized flag

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the RTC alarm flag */
```

```
rtc_flag_clear (RTC_FLAG_ALARM);
```

## 3.20. SDIO

The secure digital input/output interface (SDIO) defines the SD/SD I/O /MMC CE-ATA card host interface, which provides command/data transfer between the AHB system bus and SD memory cards, SD I/O cards, Multimedia Card (MMC), and CE-ATA devices. The SDIO registers are listed in chapter [3.20.1](#), the SDIO firmware functions are introduced in chapter [3.20.2](#).

### 3.20.1. Descriptions of Peripheral registers

SDIO registers are listed in the table shown as below:

**Table 3-444. SDIO Registers**

Registers	Descriptions
SDIO_PWRCTL	Power control register
SDIO_CLKCTL	Clock control register
SDIO_CMDAGMT	Command argument register
SDIO_CMDCTL	Command control register
SDIO_RSPCMDIDX	Command index response register
SDIO_RESPx x=0..3	Response register
SDIO_DATATO	Data timeout register
SDIO_DATALEN	Data length register
SDIO_DATACTL	Data control register
SDIO_DATACNT	Data counter register



Registers	Descriptions
SDIO_STAT	Status register
SDIO_INTC	Interrupt clear register
SDIO_INTEN	Interrupt enable register
SDIO_FIFOCNT	FIFO counter register
SDIO_FIFO	FIFO data register

### 3.20.2. Descriptions of Peripheral functions

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

**Table 3-445. SDIO firmware function**

Function name	Function description
sdio_deinit	deinitialize the SDIO
sdio_clock_config	configure the SDIO clock
sdio_hardware_clock_enable	enable hardware clock control
sdio_hardware_clock_disable	disable hardware clock control
sdio_bus_mode_set	set different SDIO card bus mode
sdio_power_state_set	set the SDIO power state
sdio_power_state_get	get the SDIO power state
sdio_clock_enable	enable SDIO_CLK clock output
sdio_clock_disable	disable SDIO_CLK clock output
sdio_command_response_config	configure the command and response
sdio_wait_type_set	set the command state machine wait type
sdio_csm_enable	enable the CSM(command state machine)
sdio_csm_disable	disable the CSM(command state machine)
sdio_command_index_get	get the last response command index
sdio_response_get	get the response for the last received command
sdio_data_config	configure the data timeout, data length and data block size
sdio_data_transfer_config	configure the data transfer mode and direction
sdio_dsm_enable	enable the DSM(data state machine) for data transfer

Function name	Function description
sdio_dsm_disable	disable the DSM(data state machine)
sdio_data_write	write data(one word) to the transmit FIFO
sdio_data_read	read data(one word) from the receive FIFO
sdio_data_counter_get	get the number of remaining data bytes to be transferred to card
sdio_fifo_counter_get	get the number of words remaining to be written or read from FIFO
sdio_dma_enable	enable the DMA request for SDIO
sdio_dma_disable	disable the DMA request for SDIO
sdio_flag_get	get the flags state of SDIO
sdio_flag_clear	clear the pending flags of SDIO
sdio_interrupt_enable	enable the SDIO interrupt
sdio_interrupt_disable	disable the SDIO interrupt
sdio_interrupt_flag_get	get the interrupt flags state of SDIO
sdio_interrupt_flag_clear	clear the interrupt pending flags of SDIO
sdio_readwait_enable	enable the read wait mode(SD I/O only)
sdio_readwait_disable	disable the read wait mode(SD I/O only)
sdio_stop_readwait_enable	enable the function that stop the read wait process(SD I/O only)
sdio_stop_readwait_disable	disable the function that stop the read wait process(SD I/O only)
sdio_readwait_type_set	set the read wait type(SD I/O only)
sdio_operation_enable	enable the SD I/O mode specific operation(SD I/O only)
sdio_operation_disable	disable the SD I/O mode specific operation(SD I/O only)
sdio_suspend_enable	enable the SD I/O suspend operation(SD I/O only)
sdio_suspend_disable	disable the SD I/O suspend operation(SD I/O only)
sdio_ceata_command_enable	enable the CE-ATA command(CE-ATA only)
sdio_ceata_command_disable	disable the CE-ATA command(CE-ATA only)
sdio_ceata_interrupt_enable	enable the CE-ATA interrupt(CE-ATA only)

Function name	Function description
sdio_ceata_interrupt_disable	disable the CE-ATA interrupt(CE-ATA only)
sdio_ceata_command_completion_enable	enable the CE-ATA command completion signal(CE-ATA only)
sdio_ceata_command_completion_disable	disable the CE-ATA command completion signal(CE-ATA only)
sdio_clock_config	configure the SDIO clock

### sdio\_deinit

The description of sdio\_deinit is shown as below:

**Table 3-446. Function sdio\_deinit**

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

Example:

```
/* deinitialize the SDIO */
sdio_deinit();
```

### sdio\_clock\_config

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

**Table 3-447. Function sdio\_clock\_config**

Function name	sdio_clock_config
---------------	-------------------

<b>Function prototype</b>	void sdio_clock_config(uint32_t clock_edge, uint32_t clock_bypass, uint32_t clock_powersave, uint16_t clock_division);
<b>Function descriptions</b>	configure the SDIO clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>clock_edge</b>	SDIO_CLK clock edge
<i>SDIO_SDIOLCKEDGE_RISING</i>	select the rising edge of the SDIOCLK to generate SDIO_CLK
<i>SDIO_SDIOLCKEDGE_FALLING</i>	select the falling edge of the SDIOCLK to generate SDIO_CLK
<b>Input parameter{in}</b>	
<b>clock_bypass</b>	clock bypass
<i>SDIO_CLOCKBYPASS_ENABLE</i>	clock bypass
<i>SDIO_CLOCKBYPASS_DISABLE</i>	no bypass
<b>Input parameter{in}</b>	
<b>clock_powersave</b>	SDIO_CLK clock dynamic switch on/off for power saving
<i>SDIO_CLOCKPWRSA_VE_ENABLE</i>	SDIO_CLK closed when bus is idle
<i>SDIO_CLOCKPWRSA_VE_DISABLE</i>	SDIO_CLK clock is always on
<b>Input parameter{in}</b>	
<b>clock_division</b>	clock division, less than 512
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the SDIO clock */
```

```
sdio_clock_config(SDIO_SDIOLCKEDGE_RISING, SDIO_CLOCKBYPASS_DISABLE,
SDIO_CLOCKPWRSAVE_DISABLE, SD_CLK_DIV_TRANS);
```

### sdio\_hardware\_clock\_enable

The description of sdio\_hardware\_clock\_enable is shown as below:

**Table 3-448. Function sdio\_hardware\_clock\_enable**

<b>Function name</b>	sdio_hardware_clock_enable
<b>Function prototype</b>	void sdio_hardware_clock_enable(void);
<b>Function descriptions</b>	enable hardware clock control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable hardware clock control */
sdio_hardware_clock_enable();
```

### sdio\_hardware\_clock\_disable

The description of sdio\_hardware\_clock\_disable is shown as below:

**Table 3-449. Function sdio\_hardware\_clock\_disable**

<b>Function name</b>	sdio_hardware_clock_disable
<b>Function prototype</b>	void sdio_hardware_clock_disable(void);
<b>Function descriptions</b>	disable hardware clock control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

```
/* disable hardware clock control */
```

```
sdio_hardware_clock_disable();
```

### sdio\_bus\_mode\_set

The description of sdio\_bus\_mode\_set is shown as below:

**Table 3-450. Function sdio\_bus\_mode\_set**

<b>Function name</b>	sdio_bus_mode_set
<b>Function prototype</b>	void sdio_bus_mode_set(uint32_t bus_mode);
<b>Function descriptions</b>	set different SDIO card bus mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>bus_mode</b>	SDIO card bus mode
<i>SDIO_BUSMODE_1BIT</i>	1-bit SDIO card bus mode
<i>SDIO_BUSMODE_4BIT</i>	4-bit SDIO card bus mode
<i>SDIO_BUSMODE_8BIT</i>	8-bit SDIO card bus mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set SDIO bus mode */
```

```
sdio_bus_mode_set(SDIO_BUSMODE_1BIT);
```

### sdio\_power\_state\_set

The description of sdio\_power\_state\_set is shown as below:

**Table 3-451. Function sdio\_power\_state\_set**

<b>Function name</b>	sdio_power_state_set
<b>Function prototype</b>	void sdio_power_state_set(uint32_t power_state);
<b>Function descriptions</b>	set the SDIO power state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>power_state</b>	SDIO power state
SDIO_POWER_ON	SDIO power on
SDIO_POWER_OFF	SDIO power off
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set SDIO power state */
```

```
sdio_power_state_set(SDIO_POWER_ON);
```

### sdio\_power\_state\_get

The description of sdio\_power\_state\_get is shown as below:

**Table 3-452. Function sdio\_power\_state\_get**

<b>Function name</b>	sdio_power_state_get
<b>Function prototype</b>	uint32_t sdio_power_state_get(void);
<b>Function descriptions</b>	get the SDIO power state
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	SDIO_POWER_ON / SDIO_POWER_OFF

Example:

```

/* get the SDIO power state */

uint32_t sdio_power_value;

sdio_power_value = sdio_power_state_get();

```

### sdio\_clock\_enable

The description of sdio\_clock\_enable is shown as below:

**Table 3-453. Function sdio\_clock\_enable**

<b>Function name</b>	sdio_clock_enable
<b>Function prototype</b>	void sdio_clock_enable(void);
<b>Function descriptions</b>	enable SDIO_CLK clock output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* enable SDIO_CLK clock output */

sdio_clock_enable();

```



## sdio\_clock\_disable

The description of sdio\_clock\_disable is shown as below:

**Table 3-454. Function sdio\_clock\_disable**

<b>Function name</b>	sdio_clock_disable
<b>Function prototype</b>	void sdio_clock_disable(void);
<b>Function descriptions</b>	disable SDIO_CLK clock output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SDIO_CLK clock output */
sdio_clock_disable();
```

## sdio\_command\_response\_config

The description of sdio\_command\_response\_config is shown as below:

**Table 3-455. Function sdio\_command\_response\_config**

<b>Function name</b>	sdio_command_response_config
<b>Function prototype</b>	void sdio_command_response_config(uint32_t cmd_index, uint32_t cmd_argument, uint32_t response_type);
<b>Function descriptions</b>	configure the command and response
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmd_index</b>	command index, refer to the related specifications

Input parameter{in}	
<b>cmd_argument</b>	command argument, refer to the related specifications
Input parameter{in}	
<b>response_type</b>	response type
<i>SDIO_RESPONSETYP</i> <i>E_NO</i>	no response
<i>SDIO_RESPONSETYP</i> <i>E_SHORT</i>	short response
<i>SDIO_RESPONSETYP</i> <i>E_LONG</i>	long response
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CMD2(SD_CMD_ALL_SEND_CID) command response config*/
```

```
sdio_command_response_config(SD_CMD_ALL_SEND_CID, (uint32_t)0x0,
SDIO_RESPONSETYPE_LONG);
```

### sdio\_wait\_type\_set

The description of sdio\_wait\_type\_set is shown as below:

**Table 3-456. Function sdio\_wait\_type\_set**

<b>Function name</b>	sdio_wait_type_set
<b>Function prototype</b>	void sdio_wait_type_set(uint32_t wait_type);
<b>Function descriptions</b>	set the command state machine wait type
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>wait_type</b>	wait type
<i>SDIO_WAITTYPE_NO</i>	not wait interrupt

<i>SDIO_WAITTYPE_INT ERRUPT</i>	wait interrupt
<i>SDIO_WAITTYPE_DATA AEND</i>	wait the end of data transfer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the command state machine wait type */
sdio_wait_type_set(SDIO_WAITTYPE_NO);
```

### sdio\_csm\_enable

The description of sdio\_csm\_enable is shown as below:

**Table 3-457. Function sdio\_csm\_enable**

<b>Function name</b>	sdio_csm_enable
<b>Function prototype</b>	void sdio_csm_enable(void);
<b>Function descriptions</b>	enable the CSM(command state machine)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the CSM(command state machine) */
sdio_csm_enable();
```

## sdio\_csm\_disable

The description of sdio\_csm\_disable is shown as below:

**Table 3-458. Function sdio\_csm\_disable**

<b>Function name</b>	sdio_csm_disable
<b>Function prototype</b>	void sdio_csm_disable(void);
<b>Function descriptions</b>	disable the CSM(command state machine)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the CSM(command state machine) */
sdio_csm_disable();
```

## sdio\_command\_index\_get

The description of sdio\_command\_index\_get is shown as below:

**Table 3-459. Function sdio\_command\_index\_get**

<b>Function name</b>	sdio_command_index_get
<b>Function prototype</b>	uint8_t sdio_command_index_get(void);
<b>Function descriptions</b>	get the last response command index
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	

-	-
<b>Return value</b>	
<b>uint8_t</b>	last response command index

Example:

```
/* get SDIO command index */
uint8_t sdio_commond_value;
sdio_commond_value = sdio_command_index_get();
```

### sdio\_response\_get

The description of sdio\_response\_get is shown as below:

**Table 3-460. Function sdio\_response\_get**

<b>Function name</b>	sdio_response_get
<b>Function prototype</b>	uint32_t sdio_response_get(uint32_t responsex);
<b>Function descriptions</b>	get the response for the last received command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>responsex</b>	SDIO response
<i>SDIO_RESPONSE0</i>	card response[31:0]/card response[127:96]
<i>SDIO_RESPONSE1</i>	card response[95:64]
<i>SDIO_RESPONSE2</i>	card response[63:32]
<i>SDIO_RESPONSE3</i>	card response[31:1], plus bit 0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	response for the last received command

Example:

```
/* store the CID0 numbers */
uint32_t sdio_cid[0];
```

```
sdio_cid[0] = sdio_response_get(SDIO_RESPONSE0);
```

## sdio\_data\_config

The description of sdio\_data\_config is shown as below:

**Table 3-461. Function sdio\_data\_config**

<b>Function name</b>	sdio_data_config
<b>Function prototype</b>	void sdio_data_config(uint32_t data_timeout, uint32_t data_length, uint32_t data_blocksize);
<b>Function descriptions</b>	configure the data timeout, data length and data block size
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data_timeout</b>	data timeout period in card bus clock periods
<b>Input parameter{in}</b>	
<b>data_length</b>	number of data bytes to be transferred
<b>Input parameter{in}</b>	
<b>data_blocksize</b>	size of data block for block transfer
SDIO_DATABLOCKSIZE_1BYTE	block size = 1 byte
SDIO_DATABLOCKSIZE_2BYTES	block size = 2 bytes
SDIO_DATABLOCKSIZE_4BYTES	block size = 4 bytes
SDIO_DATABLOCKSIZE_8BYTES	block size = 8 bytes
SDIO_DATABLOCKSIZE_16BYTES	block size = 16 bytes
SDIO_DATABLOCKSIZE_32BYTES	block size = 32 bytes
SDIO_DATABLOCKSIZE_64BYTES	block size = 64 bytes
SDIO_DATABLOCKSIZE_128BYTES	block size = 128 bytes

<i>E_128BYTES</i>	
<i>SDIO_DATABLOCKSIZE</i> <i>E_256BYTES</i>	block size = 256 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_512BYTES</i>	block size = 512 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_1024BYTES</i>	block size = 1024 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_2048BYTES</i>	block size = 2048 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_4096BYTES</i>	block size = 4096 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_8192BYTES</i>	block size = 8192 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_16384BYTES</i>	block size = 16384 bytes
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SDIO data */
```

```
sdio_data_config(0, 0, SDIO_DATABLOCKSIZE_1BYTE);
```

### **sdio\_data\_transfer\_config**

The description of `sdio_data_transfer_config` is shown as below:

**Table 3-462. Function `sdio_data_transfer_config`**

<b>Function name</b>	<code>sdio_data_transfer_config</code>
<b>Function prototype</b>	<code>void sdio_data_transfer_config(uint32_t transfer_mode, uint32_t transfer_direction);</code>
<b>Function descriptions</b>	configure the data transfer mode and direction
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>transfer_mode</b>	mode of data transfer
<i>SDIO_TRANSMODE_BLOCK</i>	block transfer
<i>SDIO_TRANSMODE_STREAM</i>	stream transfer or SDIO multibyte transfer
Input parameter{in}	
<b>transfer_direction</b>	data transfer direction, read or write
<i>SDIO_TRANSDIRECTION_TOCARD</i>	write data to card
<i>SDIO_TRANSDIRECTION_TOSDIO</i>	read data from card
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SDIO data transmisson */
```

```
sdio_data_transfer_config(SDIO_TRANSDIRECTION_TOSDIO,
SDIO_TRANSMODE_BLOCK);
```

### sdio\_dsm\_enable

The description of sdio\_dsm\_enable is shown as below:

**Table 3-463. Function sdio\_dsm\_enable**

<b>Function name</b>	sdio_dsm_enable
<b>Function prototype</b>	void sdio_dsm_enable(void);
<b>Function descriptions</b>	enable the DSM(data state machine) for data transfer
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-



Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the DSM(data state machine) */
```

```
sdio_dsm_enable();
```

### sdio\_dsm\_disable

The description of sdio\_dsm\_disable is shown as below:

**Table 3-464. Function sdio\_dsm\_disable**

Function name	sdio_dsm_disable
Function prototype	void sdio_dsm_disable(void);
Function descriptions	disable the DSM(data state machine)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the DSM(data state machine) */
```

```
sdio_dsm_disable();
```

### sdio\_data\_write

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

**Table 3-465. Function sdio\_data\_write**

Function name	sdio_data_write
---------------	-----------------

<b>Function prototype</b>	void sdio_data_write(uint32_t data);
<b>Function descriptions</b>	write data(one word) to the transmit FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data</b>	32-bit data write to card
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* write data(one word) to the transmit FIFO */
```

```
sdio_data_write(0x0000 0001);
```

### sdio\_data\_read

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

**Table 3-466. Function sdio\_data\_read**

<b>Function name</b>	sdio_data_read
<b>Function prototype</b>	uint32_t sdio_data_read(void);
<b>Function descriptions</b>	read data(one word) from the receive FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	received data

Example:

```
/* read data(one word) from the receive FIFO */
```

```
sdio_data_read();
```

### sdio\_data\_counter\_get

The description of sdio\_data\_counter\_get is shown as below:

**Table 3-467. Function sdio\_data\_counter\_get**

<b>Function name</b>	sdio_data_counter_get
<b>Function prototype</b>	uint32_t sdio_data_counter_get(void);
<b>Function descriptions</b>	get the number of remaining data bytes to be transferred to card
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	number of remaining data bytes to be transferred

Example:

```
/* get the number of remaining data bytes to be transferred to card */
```

```
uint32_t sdio_data_value;
```

```
sdio_data_value = sdio_data_counter_get();
```

### sdio\_fifo\_counter\_get

The description of sdio\_fifo\_counter\_get is shown as below:

**Table 3-468. Function sdio\_data\_counter\_get**

<b>Function name</b>	sdio_fifo_counter_get
<b>Function prototype</b>	uint32_t sdio_fifo_counter_get(void);
<b>Function descriptions</b>	get the number of words remaining to be written or read from FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	remaining number of words

Example:

```
/* get the number of words remaining to be written or read from FIFO */
```

```
uint32_t sdio_fifo_value;
```

```
sdio_fifo_value = sdio_fifo_counter_get();
```

### sdio\_dma\_enable

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

**Table 3-469. Function sdio\_dma\_enable**

Function name	sdio_dma_enable
Function prototype	void sdio_dma_enable(void);
Function descriptions	enable the DMA request for SDIO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the SDIO DMA */
```

```
sdio_dma_enable();
```

## sdio\_dma\_disable

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

**Table 3-470. Function sdio\_dma\_disable**

<b>Function name</b>	sdio_dma_disable
<b>Function prototype</b>	void sdio_dma_disable(void);
<b>Function descriptions</b>	disable the DMA request for SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the SDIO DMA */
sdio_dma_disable();
```

## sdio\_flag\_get

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

**Table 3-471. Function sdio\_flag\_get**

<b>Function name</b>	sdio_flag_get
<b>Function prototype</b>	FlagStatus sdio_flag_get(uint32_t flag);
<b>Function descriptions</b>	get the flags state of SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flags state of SDIO
<i>SDIO_FLAG_CCRCER</i>	command response received (CRC check failed) flag

<i>R</i>	
<i>SDIO_FLAG_DTCRCE</i> <i>RR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_FLAG_CMDTMO</i> <i>UT</i>	command response timeout flag
<i>SDIO_FLAG_DTTMOU</i> <i>T</i>	data timeout flag
<i>SDIO_FLAG_TXURE</i>	transmit FIFO underrun error occurs flag
<i>SDIO_FLAG_RXORE</i>	received FIFO overrun error occurs flag
<i>SDIO_FLAG_CMDREC</i> <i>V</i>	command response received (CRC check passed) flag
<i>SDIO_FLAG_CMDSEN</i> <i>D</i>	command sent (no response required) flag
<i>SDIO_FLAG_DTEND</i>	data end (data counter, <i>SDIO_DATACNT</i> , is zero) flag
<i>SDIO_FLAG_STBITE</i>	start bit error in the bus flag
<i>SDIO_FLAG_DTBLKE</i> <i>ND</i>	data block sent/received (CRC check passed) flag
<i>SDIO_FLAG_CMDRUN</i>	command transmission in progress flag
<i>SDIO_FLAG_TXRUN</i>	data transmission in progress flag
<i>SDIO_FLAG_RXRUN</i>	data reception in progress flag
<i>SDIO_FLAG_TFH</i>	transmit FIFO is half empty flag: at least 8 words can be written into the FIFO
<i>SDIO_FLAG_RFH</i>	receive FIFO is half full flag: at least 8 words can be read in the FIFO
<i>SDIO_FLAG_TFF</i>	transmit FIFO is full flag
<i>SDIO_FLAG_RFF</i>	receive FIFO is full flag
<i>SDIO_FLAG_TFE</i>	transmit FIFO is empty flag
<i>SDIO_FLAG_RFE</i>	receive FIFO is empty flag
<i>SDIO_FLAG_TXDTVAL</i>	data is valid in transmit FIFO flag
<i>SDIO_FLAG_RXDTVA</i> <i>L</i>	data is valid in receive FIFO flag
<i>SDIO_FLAG_SDIOINT</i>	SD I/O interrupt received flag

<i>SDIO_FLAG_ATAEND</i>	CE-ATA command completion signal received (only for CMD61) flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the flags state of SDIO */
```

```
FlagStatus flag_value;
```

```
flag_value = sdio_flag_get(SDIO_FLAG_RFH);
```

### sdio\_flag\_clear

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

**Table 3-472. Function sdio\_flag\_clear**

<b>Function name</b>	sdio_flag_clear
<b>Function prototype</b>	void sdio_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear the pending flags of SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flags state of SDIO
<i>SDIO_FLAG_CCRCE</i> <i>R</i>	command response received (CRC check failed) flag
<i>SDIO_FLAG_DTCRCE</i> <i>RR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_FLAG_CMDTMO</i> <i>UT</i>	command response timeout flag
<i>SDIO_FLAG_DTTMOU</i> <i>T</i>	data timeout flag
<i>SDIO_FLAG_TXURE</i>	transmit FIFO underrun error occurs flag
<i>SDIO_FLAG_RXORE</i>	received FIFO overrun error occurs flag

<i>SDIO_FLAG_CMDREC</i> <i>V</i>	command response received (CRC check passed) flag
<i>SDIO_FLAG_CMDSEN</i> <i>D</i>	command sent (no response required) flag
<i>SDIO_FLAG_DTEND</i>	data end (data counter, <i>SDIO_DATACNT</i> , is zero) flag
<i>SDIO_FLAG_STBITE</i>	start bit error in the bus flag
<i>SDIO_FLAG_DTBKE</i> <i>ND</i>	data block sent/received (CRC check passed) flag
<i>SDIO_FLAG_SDIOINT</i>	SD I/O interrupt received flag
<i>SDIO_FLAG_ATAEND</i>	CE-ATA command completion signal received (only for CMD61) flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the pending flags of SDIO */
```

```
sdio_flag_clear(SDIO_FLAG_DTCRCERR);
```

### sdio\_interrupt\_enable

The description of `sdio_interrupt_enable` is shown as below:

**Table 3-473. Function `sdio_interrupt_enable`**

<b>Function name</b>	<code>sdio_interrupt_enable</code>
<b>Function prototype</b>	<code>void sdio_interrupt_enable(uint32_t int_flag);</code>
<b>Function descriptions</b>	enable the SDIO interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flags state of SDIO
<i>SDIO_INT_CCRERR</i>	SDIO CCRERR interrupt
<i>SDIO_INT_DTCRCER</i>	SDIO DTCRCERR interrupt



<i>R</i>	
<i>SDIO_INT_CMDTMOUT</i> <i>T</i>	SDIO CMDTMOUT interrupt
<i>SDIO_INT_DTTMOUT</i>	SDIO DTTMOUT interrupt
<i>SDIO_INT_TXURE</i>	SDIO TXURE interrupt
<i>SDIO_INT_RXORE</i>	SDIO_INT_RXORE
<i>SDIO_INT_CMDRECV</i>	SDIO CMDRECV interrupt
<i>SDIO_INT_CMDSEND</i>	SDIO CMDSEND interrupt
<i>SDIO_INT_DTEND</i>	SDIO DTEND interrupt
<i>SDIO_INT_STBITE</i>	SDIO STBITE interrupt
<i>SDIO_INT_DTBLKEND</i>	SDIO DTBLKEND interrupt
<i>SDIO_INT_CMDRUN</i>	SDIO CMDRUN interrupt
<i>SDIO_INT_TXRUN</i>	SDIO TXRUN interrupt
<i>SDIO_INT_RXRUN</i>	SDIO RXRUN interrupt
<i>SDIO_INT_TFH</i>	SDIO TFH interrupt
<i>SDIO_INT_RFH</i>	SDIO RFH interrupt
<i>SDIO_INT_TFF</i>	SDIO TFF interrupt
<i>SDIO_INT_RFF</i>	SDIO RFF interrupt
<i>SDIO_INT_TFE</i>	SDIO TFE interrupt
<i>SDIO_INT_RFE</i>	SDIO RFE interrupt
<i>SDIO_INT_TXDTVAL</i>	SDIO TXDTVAL interrupt
<i>SDIO_INT_RXDTVAL</i>	SDIO RXDTVAL interrupt
<i>SDIO_INT_SDIOINT</i>	SDIO SDIOINT interrupt
<i>SDIO_INT_ATAEND</i>	SDIO ATAEND interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the SDIO corresponding interrupts */
```

```
sdio_interrupt_enable(SDIO_INT_CCRERR | SDIO_INT_DTTMOUT | SDIO_INT_RXORE  
| SDIO_INT_DTEND | SDIO_INT_STBITE);
```

### sdio\_interrupt\_disable

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

**Table 3-474. Function sdio\_interrupt\_disable**

Function name	sdio_interrupt_disable
Function prototype	void sdio_interrupt_disable(uint32_t int_flag);
Function descriptions	disable the SDIO interrupt
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	interrupt flags state of SDIO
SDIO_INT_CCRERR	SDIO CCRERR interrupt
SDIO_INT_DTCRCERR	SDIO DTCRCERR interrupt
SDIO_INT_CMDTMOUT	SDIO CMDTMOUT interrupt
SDIO_INT_DTTMOUT	SDIO DTTMOUT interrupt
SDIO_INT_TXURE	SDIO TXURE interrupt
SDIO_INT_RXORE	SDIO_INT_RXORE
SDIO_INT_CMDRECV	SDIO CMDRECV interrupt
SDIO_INT_CMDSEND	SDIO CMDSEND interrupt
SDIO_INT_DTEND	SDIO DTEND interrupt
SDIO_INT_STBITE	SDIO STBITE interrupt
SDIO_INT_DTBLKEND	SDIO DTBLKEND interrupt
SDIO_INT_CMDRUN	SDIO CMDRUN interrupt
SDIO_INT_TXRUN	SDIO TXRUN interrupt
SDIO_INT_RXRUN	SDIO RXRUN interrupt

<i>SDIO_INT_TFH</i>	SDIO TFH interrupt
<i>SDIO_INT_RFH</i>	SDIO RFH interrupt
<i>SDIO_INT_TFF</i>	SDIO TFF interrupt
<i>SDIO_INT_RFF</i>	SDIO RFF interrupt
<i>SDIO_INT_TFE</i>	SDIO TFE interrupt
<i>SDIO_INT_RFE</i>	SDIO RFE interrupt
<i>SDIO_INT_TXDTVAL</i>	SDIO TXDTVAL interrupt
<i>SDIO_INT_RXDTVAL</i>	SDIO RXDTVAL interrupt
<i>SDIO_INT_SDIOINT</i>	SDIO SDIOINT interrupt
<i>SDIO_INT_ATAEND</i>	SDIO ATAEND interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the SDIO interrupt */
```

```
sdio_interrupt_disable(SDIO_INT_DTCRCERR);
```

### sdio\_interrupt\_flag\_get

The description of `sdio_interrupt_flag_get` is shown as below:

**Table 3-475. Function `sdio_interrupt_flag_get`**

<b>Function name</b>	<code>sdio_interrupt_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus sdio_interrupt_flag_get(uint32_t int_flag);</code>
<b>Function descriptions</b>	get the interrupt flags state of SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flags state of SDIO
<i>SDIO_INT_FLAG_CCR</i>	SDIO CCRERR interrupt flag

<i>CERR</i>	
<i>SDIO_INT_FLAG_DTCRCERR</i>	SDIO DTCRCERR interrupt flag
<i>SDIO_INT_FLAG_CMDTMOUT</i>	SDIO CMDTMOUT interrupt flag
<i>SDIO_INT_FLAG_DTTMOUT</i>	SDIO DTTMOUT interrupt flag
<i>SDIO_INT_FLAG_TXURRE</i>	SDIO TXURRE interrupt flag
<i>SDIO_INT_FLAG_RXORE</i>	SDIO_INT_RXORE flag
<i>SDIO_INT_FLAG_CMDRECV</i>	SDIO CMDRECV interrupt flag
<i>SDIO_INT_FLAG_CMDSEND</i>	SDIO CMDSEND interrupt flag
<i>SDIO_INT_FLAG_DTEEND</i>	SDIO DTEND interrupt flag
<i>SDIO_INT_FLAG_STBITE</i>	SDIO STBITE interrupt flag
<i>SDIO_INT_FLAG_DTBKEND</i>	SDIO DTBLKEND interrupt flag
<i>SDIO_INT_FLAG_CMDRUN</i>	SDIO CMDRUN interrupt flag
<i>SDIO_INT_FLAG_TXRUN</i>	SDIO TXRUN interrupt flag
<i>SDIO_INT_FLAG_RXRUN</i>	SDIO RXRUN interrupt flag
<i>SDIO_INT_FLAG_TFH</i>	SDIO TFH interrupt flag
<i>SDIO_INT_FLAG_RFH</i>	SDIO RFH interrupt flag
<i>SDIO_INT_FLAG_TFF</i>	SDIO TFF interrupt flag
<i>SDIO_INT_FLAG_RFF</i>	SDIO RFF interrupt flag
<i>SDIO_INT_FLAG_TFE</i>	SDIO TFE interrupt flag
<i>SDIO_INT_FLAG_RFE</i>	SDIO RFE interrupt flag

<i>SDIO_INT_FLAG_TXD TVAL</i>	SDIO TXDTVAL interrupt flag
<i>SDIO_INT_FLAG_RXD TVAL</i>	SDIO RXDTVAL interrupt flag
<i>SDIO_INT_FLAG_SDI OINT</i>	SDIO SDIOINT interrupt flag
<i>SDIO_INT_FLAG_ATA END</i>	SDIO ATAEND interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the interrupt flags state of SDIO */
```

```
FlagStatus flag_value;
```

```
flag_value = sdio_interrupt_flag_get(SDIO_INT_FLAG_DTEND);
```

### sdio\_interrupt\_flag\_clear

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

**Table 3-476. Function sdio\_interrupt\_flag\_clear**

<b>Function name</b>	sdio_interrupt_flag_clear
<b>Function prototype</b>	void sdio_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear the interrupt pending flags of SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flags state of SDIO
<i>SDIO_INT_FLAG_CCR CERR</i>	command response received (CRC check failed) flag
<i>SDIO_INT_FLAG_DTC RCERR</i>	data block sent/received (CRC check failed) flag

<i>SDIO_INT_FLAG_CMD TMOUT</i>	command response timeout flag
<i>SDIO_INT_FLAG_DTT MOUT</i>	data timeout flag
<i>SDIO_INT_FLAG_TXU RE</i>	transmit FIFO underrun error occurs flag
<i>SDIO_INT_FLAG_RXO RE</i>	received FIFO overrun error occurs flag
<i>SDIO_INT_FLAG_CMD RECV</i>	command response received (CRC check passed) flag
<i>SDIO_INT_FLAG_CMD SEND</i>	command sent (no response required) flag
<i>SDIO_INT_FLAG_DTE ND</i>	data end (data counter, SDIO_DATACNT, is zero) flag
<i>SDIO_INT_FLAG_STBI TE</i>	start bit error in the bus flag
<i>SDIO_INT_FLAG_DTB LKEND</i>	data block sent/received (CRC check passed) flag
<i>SDIO_INT_FLAG_SDI OINT</i>	SD I/O interrupt received flag
<i>SDIO_INT_FLAG_ATA END</i>	CE-ATA command completion signal received (only for CMD61) flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the interrupt pending flags of SDIO */
```

```
sdio_interrupt_flag_clear(SDIO_INT_FLAG_DTEND);
```

### **sdio\_readwait\_enable**

The description of sdio\_readwait\_enable is shown as below:

Table 3-477. Function sdio\_readwait\_enable

Function name	sdio_readwait_enable
Function prototype	void sdio_readwait_enable(void);
Function descriptions	enable the read wait mode(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the read wait mode(SD I/O only) */
```

```
sdio_readwait_enable();
```

### sdio\_readwait\_disable

The description of sdio\_readwait\_disable is shown as below:

Table 3-478. Function sdio\_readwait\_disable

Function name	sdio_readwait_disable
Function prototype	void sdio_readwait_disable(void);
Function descriptions	disable the read wait mode(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable the read wait mode(SD I/O only) */
```

```
sdio_readwait_disable();
```

### sdio\_stop\_readwait\_enable

The description of sdio\_stop\_readwait\_enable is shown as below:

**Table 3-479. Function sdio\_stop\_readwait\_enable**

<b>Function name</b>	sdio_stop_readwait_enable
<b>Function prototype</b>	void sdio_stop_readwait_enable(void);
<b>Function descriptions</b>	enable the function that stop the read wait process(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the function that stop the read wait process(SD I/O only) */
```

```
sdio_stop_readwait_enable();
```

### sdio\_stop\_readwait\_disable

The description of sdio\_stop\_readwait\_disable is shown as below:

**Table 3-480. Function sdio\_stop\_readwait\_disable**

<b>Function name</b>	sdio_stop_readwait_disable
<b>Function prototype</b>	void sdio_stop_readwait_disable(void);
<b>Function descriptions</b>	disable the function that stop the read wait process(SD I/O only)
<b>Precondition</b>	-



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

Example:

```
/* disable the function that stop the read wait process(SD I/O only) */
```

```
sdio_stop_readwait_disable();
```

### sdio\_readwait\_type\_set

The description of sdio\_readwait\_type\_set is shown as below:

**Table 3-481. Function sdio\_readwait\_type\_set**

Function name	sdio_readwait_type_set
Function prototype	void sdio_readwait_type_set(uint32_t readwait_type);
Function descriptions	set the read wait type(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
readwait_type	SD I/O read wait type
SDIO_READWAITTYP E_CLK	read wait control by stopping SDIO_CLK
SDIO_READWAITTYP E_DAT2	read wait control using SDIO_DAT[2]
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the read wait type(SD I/O only) */
```

```
sdio_readwait_type_set(uint32_t readwait_type);
```

### sdio\_operation\_enable

The description of sdio\_operation\_enable is shown as below:

**Table 3-482. Function sdio\_operation\_enable**

<b>Function name</b>	sdio_operation_enable
<b>Function prototype</b>	void sdio_operation_enable(void);
<b>Function descriptions</b>	enable the SD I/O mode specific operation(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the SD I/O mode specific operation(SD I/O only) */
```

```
sdio_operation_enable();
```

### sdio\_operation\_disable

The description of sdio\_operation\_disable is shown as below:

**Table 3-483. Function sdio\_operation\_disable**

<b>Function name</b>	sdio_operation_disable
<b>Function prototype</b>	void sdio_operation_disable(void);
<b>Function descriptions</b>	disable the SD I/O mode specific operation(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

```
/* disable the SD I/O mode specific operation(SD I/O only) */
```

```
void sdio_operation_disable();
```

### sdio\_suspend\_enable

The description of sdio\_suspend\_enable is shown as below:

**Table 3-484. Function sdio\_suspend\_enable**

Function name	sdio_suspend_enable
Function prototype	void sdio_suspend_enable(void);
Function descriptions	enable the SD I/O suspend operation(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the SD I/O suspend operation(SD I/O only) */
```

```
sdio_suspend_enable();
```

### sdio\_suspend\_disable

The description of sdio\_suspend\_disable is shown as below:

Table 3-485. Function sdio\_suspend\_disable

Function name	sdio_suspend_disable
Function prototype	void sdio_suspend_disable(void);
Function descriptions	disable the SD I/O suspend operation(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the SD I/O suspend operation(SD I/O only) */
sdio_suspend_disable();
```

### sdio\_ceata\_command\_enable

The description of sdio\_ceata\_command\_enable is shown as below:

Table 3-486. Function sdio\_ceata\_command\_enable

Function name	sdio_ceata_command_enable
Function prototype	void sdio_ceata_command_enable(void);
Function descriptions	enable the CE-ATA command(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable the CE-ATA command(CE-ATA only) */
```

```
sdio_ceata_command_enable();
```

### sdio\_ceata\_command\_disable

The description of sdio\_ceata\_command\_disable is shown as below:

**Table 3-487. Function sdio\_ceata\_command\_disable**

<b>Function name</b>	sdio_ceata_command_disable
<b>Function prototype</b>	void sdio_ceata_command_disable(void);
<b>Function descriptions</b>	disable the CE-ATA command(CE-ATA only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the CE-ATA command(CE-ATA only) */
```

```
sdio_ceata_command_disable();
```

### sdio\_ceata\_interrupt\_enable

The description of sdio\_ceata\_interrupt\_enable is shown as below:

**Table 3-488. Function sdio\_ceata\_interrupt\_enable**

<b>Function name</b>	sdio_ceata_interrupt_enable
<b>Function prototype</b>	void sdio_ceata_interrupt_enable(void);
<b>Function descriptions</b>	enable the CE-ATA interrupt(CE-ATA only)
<b>Precondition</b>	-

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

Example:

```
/* enable the CE-ATA interrupt(CE-ATA only) */
```

```
sdio_ceata_interrupt_enable();
```

### **sdio\_ceata\_interrupt\_disable**

The description of sdio\_ceata\_interrupt\_disable is shown as below:

**Table 3-489. Function sdio\_ceata\_interrupt\_disable**

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

Example:

```
/* disable the CE-ATA interrupt(CE-ATA only) */
```

```
sdio_ceata_interrupt_disable();
```

## sdio\_ceata\_command\_completion\_enable

The description of sdio\_ceata\_command\_completion\_enable is shown as below:

**Table 3-490. Function sdio\_ceata\_command\_completion\_enable**

<b>Function name</b>	sdio_ceata_command_completion_enable
<b>Function prototype</b>	void sdio_ceata_command_completion_enable(void);
<b>Function descriptions</b>	enable the CE-ATA command completion signal(CE-ATA only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the CE-ATA command completion signal(CE-ATA only) */
sdio_ceata_command_completion_enable();
```

## sdio\_ceata\_command\_completion\_disable

The description of sdio\_ceata\_command\_completion\_disable is shown as below:

**Table 3-491. Function sdio\_ceata\_command\_completion\_disable**

<b>Function name</b>	sdio_ceata_command_completion_disable
<b>Function prototype</b>	void sdio_ceata_command_completion_disable(void);
<b>Function descriptions</b>	disable the CE-ATA command completion signal(CE-ATA only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* disable the CE-ATA command completion signal(CE-ATA only) */
sdio_ceata_command_completion_disable();
```

## 3.21. SPI

The SPI/I2S module can communicate with external devices using the SPI protocol or the I2S audio protocol. The SPI/I2S registers are listed in chapter [3.21.1](#), the SPI/I2S firmware functions are introduced in chapter [3.21.2](#).

### 3.21.1. Descriptions of Peripheral registers

SPI/I2S registers are listed in the table shown as below:

**Table 3-492. SPI/I2S Registers**

Registers	Descriptions
SPI_CTL0	SPI control register 0
SPI_CTL1	SPI control register 1
SPI_STAT	SPI status register
SPI_DATA	SPI data register
SPI_CRCPOLY	SPI CRC polynomial register
SPI_RCRC	SPI receive CRC register
SPI_TCRC	SPI transmit CRC register
SPI_I2SCTL	I2S control register
SPI_I2SPSC	I2S clock prescaler register
SPI_QCTL	Quad-SPI mode control register

### 3.21.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:



Table 3-493. SPI/I2S firmware function

Function name	Function description
spi_i2s_deinit	reset SPI and I2S peripheral
spi_struct_para_init	initialize the parameters of SPI structure with the default values
spi_init	initialize SPI peripheral parameter
spi_enable	enable SPI
spi_disable	disable SPI
i2s_init	initialize I2S peripheral parameters
i2s_psc_config	configure I2S peripheral prescaler
i2s_enable	enable I2S
i2s_disable	disable I2S
spi_nss_output_enable	enable SPI NSS output function
spi_nss_output_disable	disable SPI NSS output function
spi_nss_internal_high	SPI NSS pin high level in software mode
spi_nss_internal_low	SPI NSS pin low level in software mode
spi_dma_enable	enable SPI DMA send or receive
spi_dma_disable	disable SPI DMA send or receive
spi_i2s_data_frame_format_config	configure SPI/I2S data frame format
spi_i2s_data_transmit	SPI transmit data
spi_i2s_data_receive	SPI receive data
spi_bidirectional_transfer_config	configure SPI bidirectional transfer direction
spi_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_ti_mode_enable	enable SPI TI mode

Function name	Function description
spi_ti_mode_disable	disable SPI TI mode
spi_nssp_mode_enable	enable SPI NSS pulse mode
spi_nssp_mode_disable	disable SPI NSS pulse mode
spi_quad_enable	enable quad wire SPI
spi_quad_disable	disable quad wire SPI
spi_quad_write_enable	enable quad wire SPI write
spi_quad_read_enable	enable quad wire SPI read
spi_quad_io23_output_enable	enable quad wire SPI_IO2 and SPI_IO3 pin output
spi_quad_io23_output_disable	disable quad wire SPI_IO2 and SPI_IO3 pin output
spi_i2s_interrupt_enable	enable SPI and I2S interrupt
spi_i2s_interrupt_disable	disable SPI and I2S interrupt
spi_i2s_interrupt_flag_get	get SPI and I2S interrupt status
spi_i2s_flag_get	get SPI and I2S flag status
spi_crc_error_clear	clear SPI CRC error flag status

## Structure spi\_parameter\_struct

Table 3-494. spi\_parameter\_struct

Member name	Function description
device_mode	SPI master or slave (SPI_MASTER, SPI_SLAVE)
trans_mode	SPI transfer type (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	SPI frame size (SPI_FRAME_SIZE_16BIT, SPI_FRAME_SIZE_8BIT)
nss	SPI NSS control by hardware or software (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	SPI big endian or little endian

	(SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	SPI clock phase and polarity (SPI_CK_PL_LOW_PH_1EDGE, SPI_CK_PL_HIGH_PH_1EDGE, SPI_CK_PL_LOW_PH_2EDGE, SPI_CK_PL_HIGH_PH_2EDGE)
prescale	SPI prescale factor (SPI_PSC_n (n=2,4,8,16,32,64,128,256))

### spi\_i2s\_deinit

The description of spi\_i2s\_deinit is shown as below:

**Table 3-495. Function spi\_i2s\_deinit**

<b>Function name</b>	spi_i2s_deinit
<b>Function prototype</b>	void spi_i2s_deinit(uint32_t spi_periph);
<b>Function descriptions</b>	reset SPI and I2S
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI/I2S peripheral
<b>SPIx</b>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset SPI0 */
spi_i2s_deinit(SPI0);
```

### spi\_struct\_para\_init

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

**Table 3-496. 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 structure with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
spi_struct	SPI init parameter structure, the structure members can refer to <a href="#">Table 3-494. spi parameter struct</a>
<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 3-497. Function spi\_init**

<b>Function name</b>	spi_init
<b>Function prototype</b>	void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
<b>Function descriptions</b>	initialize SPI peripheral parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1,2
<b>Input parameter{in}</b>	
spi_struct	SPI parameter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-494. spi parameter struct</a>

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize SPI0 */
```

```
spi_parameter_struct spi_init_struct;
```

```
spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
```

```
spi_init_struct.device_mode     = SPI_MASTER;
```

```
spi_init_struct.frame_size      = SPI_FRAME_SIZE_8BIT;
```

```
spi_init_struct.clock_polarity_phase = SPI_CK_PL_HIGH_PH_2EDGE;
```

```
spi_init_struct.nss             = SPI_NSS_SOFT;
```

```
spi_init_struct.prescale        = SPI_PSC_8;
```

```
spi_init_struct.endian          = SPI_ENDIAN_MSB;
```

```
spi_init(SPI0, &spi_init_struct);
```

## spi\_enable

The description of spi\_enable is shown as below:

**Table 3-498. Function spi\_enable**

Function name	spi_enable
Function prototype	void spi_enable(uint32_t spi_periph);
Function descriptions	enable SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* enable SPI0 */
```

```
spi_enable(SPI0);
```

### spi\_disable

The description of spi\_disable is shown as below:

**Table 3-499. Function spi\_disable**

Function name	spi_disable
Function prototype	void spi_disable(uint32_t spi_periph);
Function descriptions	disable SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 */
```

```
spi_disable(SPI0);
```

### i2s\_init

The description of i2s\_init is shown as below:

**Table 3-500. Function i2s\_init**

Function name	i2s_init
Function prototype	void i2s_init(uint32_t spi_periph, uint32_t i2s_mode, uint32_t i2s_standard,

	uint32_t i2s_ckpl);
<b>Function descriptions</b>	initialize I2S peripheral parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<i>SPIx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>i2s_mode</b>	I2S operation mode
<i>I2S_MODE_SLAVETX</i>	I2S slave transmit mode
<i>I2S_MODE_SLAVERX</i>	I2S slave receive mode
<i>I2S_MODE_MASTERTX</i>	I2S master transmit mode
<i>I2S_MODE_MASTERRX</i>	I2S master receive mode
<b>Input parameter{in}</b>	
<b>i2s_standard</b>	I2S standard
<i>I2S_STD_PHILLIPS</i>	I2S phillips standard
<i>I2S_STD_MSB</i>	I2S MSB standard
<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCMSHORT</i>	I2S PCM short standard
<i>I2S_STD_PCMLONG</i>	I2S PCM long standard
<b>Input parameter{in}</b>	
<b>i2s_ckpl</b>	I2S idle state clock polarity
<i>I2S_CKPL_LOW</i>	I2S clock polarity low level
<i>I2S_CKPL_HIGH</i>	I2S clock polarity high level
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* initialize I2S1 */
```

```
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

### i2s\_psc\_config

The description of i2s\_psc\_config is shown as below:

**Table 3-501. Function i2s\_psc\_config**

<b>Function name</b>	i2s_psc_config
<b>Function prototype</b>	void i2s_psc_config(uint32_t spi_periph, uint32_t i2s_audiosample, uint32_t i2s_frameformat, uint32_t i2s_mckout);
<b>Function descriptions</b>	configure I2S prescaler
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<i>SPIx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>i2s_audiosample</b>	I2S audio sample rate
<i>I2S_AUDIOSAMPLE_8K</i>	audio sample rate is 8KHz
<i>I2S_AUDIOSAMPLE_11K</i>	audio sample rate is 11KHz
<i>I2S_AUDIOSAMPLE_16K</i>	audio sample rate is 16KHz
<i>I2S_AUDIOSAMPLE_22K</i>	audio sample rate is 22KHz
<i>I2S_AUDIOSAMPLE_32K</i>	audio sample rate is 32KHz
<i>I2S_AUDIOSAMPLE_44K</i>	audio sample rate is 44KHz



<i>I2S_AUDIOSAMPLE_4</i> 8K	audio sample rate is 48KHz
<i>I2S_AUDIOSAMPLE_9</i> 6K	audio sample rate is 96KHz
<i>I2S_AUDIOSAMPLE_1</i> 92K	audio sample rate is 192KHz
<b>Input parameter{in}</b>	
<b>i2s_frameformat</b>	I2S data length and channel length
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S data length is 16 bit and channel length is 16 bit
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S data length is 16 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S data length is 24 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S data length is 32 bit and channel length is 32 bit
<b>Input parameter{in}</b>	
<b>i2s_mckout</b>	I2S master clock output
<i>I2S_MCKOUT_ENABL</i> <i>E</i>	I2S master clock output enable
<i>I2S_MCKOUT_DISABL</i> <i>E</i>	I2S master clock output disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2S1 prescaler */
```

```
i2s_psc_config(SPI1, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,
I2S_MCKOUT_DISABLE);
```

## i2s\_enable

The description of i2s\_enable is shown as below:

**Table 3-502. Function i2s\_enable**

<b>Function name</b>	i2s_enable
<b>Function prototype</b>	void i2s_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable I2S
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<i>SPIx</i>	x=1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2S1*/
i2s_enable(SPI1);
```

## i2s\_disable

The description of i2s\_disable is shown as below:

**Table 3-503. Function i2s\_disable**

<b>Function name</b>	i2s_disable
<b>Function prototype</b>	void i2s_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable I2S
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral

<i>SPI</i> <i>x</i>	<i>x</i> =1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2S1*/
```

```
i2s_disable(SPI1);
```

### **spi\_nss\_output\_enable**

The description of spi\_nss\_output\_enable is shown as below:

**Table 3-504. Function spi\_nss\_output\_enable**

<b>Function name</b>	spi_nss_output_enable
<b>Function prototype</b>	void spi_nss_output_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI NSS output
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>spi_periph</b>	SPI peripheral
<i>SPI</i> <i>x</i>	<i>x</i> =0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 NSS output */
```

```
spi_nss_output_enable(SPI0);
```

### **spi\_nss\_output\_disable**

The description of spi\_nss\_output\_disable is shown as below:

Table 3-505. Function spi\_nss\_output\_disable

Function name	spi_nss_output_disable
Function prototype	void spi_nss_output_disable(uint32_t spi_periph);
Function descriptions	disable SPI NSS output
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 NSS output */
spi_nss_output_disable(SPI0);
```

### spi\_nss\_internal\_high

The description of spi\_nss\_internal\_high is shown as below:

Table 3-506. Function spi\_nss\_internal\_high

Function name	spi_nss_internal_high
Function prototype	void spi_nss_internal_high(uint32_t spi_periph);
Function descriptions	SPI NSS pin high level in software mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* SPI0 NSS pin is pulled high level in software mode */
```

```
spi_nss_internal_high(SPI0);
```

### spi\_nss\_internal\_low

The description of spi\_nss\_internal\_low is shown as below:

**Table 3-507. Function spi\_nss\_internal\_low**

Function name	spi_nss_internal_low
Function prototype	void spi_nss_internal_low(uint32_t spi_periph);
Function descriptions	SPI NSS pin low level in software mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low(SPI0);
```

### spi\_dma\_enable

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

**Table 3-508. Function spi\_dma\_enable**

Function name	spi_dma_enable
---------------	----------------

<b>Function prototype</b>	void spi_dma_enable(uint32_t spi_periph, uint8_t spi_dma);
<b>Function descriptions</b>	enable SPI DMA send or receive
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>spi_dma</b>	SPI DMA mode
<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA
<i>SPI_DMA_RECEIVE</i>	SPI receive data use DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 transmit data DMA function */
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

### spi\_dma\_disable

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

**Table 3-509. Function spi\_dma\_disable**

<b>Function name</b>	spi_dma_disable
<b>Function prototype</b>	void spi_dma_disable(uint32_t spi_periph, uint8_t spi_dma);
<b>Function descriptions</b>	disable SPI DMA send or receive
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral

<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>spi_dma</b>	SPI DMA mode
<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA
<i>SPI_DMA_RECEIVE</i>	SPI receive data use DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 transmit data DMA function */
```

```
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

### spi\_i2s\_data\_frame\_format\_config

The description of spi\_i2s\_data\_frame\_format\_config is shown as below:

**Table 3-510. Function spi\_i2s\_data\_frame\_format\_config**

<b>Function name</b>	spi_i2s_data_frame_format_config
<b>Function prototype</b>	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
<b>Function descriptions</b>	configure SPI/I2S data frame format
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>frame_format</b>	SPI frame size
<i>SPI_FRAME_SIZE_16BIT</i>	SPI frame size is 16 bits
<i>SPI_FRAME_SIZE_8BIT</i>	SPI frame size is 8 bits

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SPI1/I2S1 data frame format size is 16 bits */
```

```
spi_i2s_data_frame_format_config(SPI1, SPI_FRAMESIZE_16BIT);
```

### spi\_i2s\_data\_transmit

The description of spi\_i2s\_data\_transmit is shown as below:

**Table 3-511. Function spi\_i2s\_data\_transmit**

Function name	spi_i2s_data_transmit
Function prototype	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
Function descriptions	SPI transmit data
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Input parameter{in}	
data	16-bit data
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 transmit data */
```

```
uint16_t spi0_send_array[] = {0x5050, 0xA0A0};
```

```
spi_i2s_data_transmit(SPI0, spi0_send_array[0]);
```



## spi\_i2s\_data\_receive

The description of spi\_i2s\_data\_receive is shown as below:

**Table 3-512. Function spi\_i2s\_data\_receive**

<b>Function name</b>	spi_i2s_data_receive
<b>Function prototype</b>	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
<b>Function descriptions</b>	SPI receive data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	16-bit data

Example:

```
/* SPI0 receive data */
uint16_t spi0_receive_data;
spi0_receive_data = spi_i2s_data_receive(SPI0);
```

## spi\_bidirectional\_transfer\_config

The description of spi\_bidirectional\_transfer\_config is shown as below:

**Table 3-513. Function spi\_bidirectional\_transfer\_config**

<b>Function name</b>	spi_bidirectional_transfer_config
<b>Function prototype</b>	void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);
<b>Function descriptions</b>	configure SPI bidirectional transfer direction
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
Input parameter{in}	
<b>transfer_direction</b>	SPI transfer direction
<i>SPI_BIDIRECTIONAL_TRANSMIT</i>	SPI work in transmit-only mode
<i>SPI_BIDIRECTIONAL_RECEIVE</i>	SPI work in receive-only mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

### spi\_crc\_polynomial\_set

The description of spi\_crc\_polynomial\_set is shown as below:

**Table 3-514. Function spi\_crc\_polynomial\_set**

<b>Function name</b>	spi_crc_polynomial_set
<b>Function prototype</b>	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);
<b>Function descriptions</b>	set SPI CRC polynomial
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
Input parameter{in}	
<b>crc_poly</b>	CRC polynomial value

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set SPI0 CRC polynomial */
uint16_t CRC_VALUE = 0x55AA;
spi_crc_polynomial_set(SPI0,CRC_VALUE);
```

### spi\_crc\_polynomial\_get

The description of spi\_crc\_polynomial\_get is shown as below:

**Table 3-515. Function spi\_crc\_polynomial\_get**

Function name	spi_crc_polynomial_get
Function prototype	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
Function descriptions	get SPI CRC polynomial
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
uint16_t	16-bit CRC polynomial (0-0xFFFF)

Example:

```
/* get SPI0 CRC polynomial */
uint16_t crc_val;
crc_val = spi_crc_polynomial_get(SPI0);
```

## spi\_crc\_on

The description of spi\_crc\_on is shown as below:

**Table 3-516. Function spi\_crc\_on**

<b>Function name</b>	spi_crc_on
<b>Function prototype</b>	void spi_crc_on(uint32_t spi_periph);
<b>Function descriptions</b>	turn on CRC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn on SPI0 CRC function */
```

```
spi_crc_on(SPI0);
```

## spi\_crc\_off

The description of spi\_crc\_off is shown as below:

**Table 3-517. Function spi\_crc\_off**

<b>Function name</b>	spi_crc_off
<b>Function prototype</b>	void spi_crc_off(uint32_t spi_periph);
<b>Function descriptions</b>	turn off CRC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral

<i>SPI</i> <i>x</i>	<i>x</i> =0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn off SPI0 CRC function */
```

```
spi_crc_off(SPI0);
```

### spi\_crc\_next

The description of spi\_crc\_next is shown as below:

**Table 3-518. Function spi\_crc\_next**

Function name	spi_crc_next
Function prototype	void spi_crc_next(uint32_t spi_periph);
Function descriptions	SPI next data is CRC value
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPI</i> <i>x</i>	<i>x</i> =0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 next data is CRC value */
```

```
spi_crc_next(SPI0);
```

### spi\_crc\_get

The description of spi\_crc\_get is shown as below:

Table 3-519. Function spi\_crc\_get

Function name	spi_crc_get
Function prototype	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc);
Function descriptions	get SPI CRC send value or receive value
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Input parameter{in}	
crc	SPI crc value
SPI_CRC_TX	get transmit crc value
SPI_CRC_RX	get receive crc value
Output parameter{out}	
-	-
Return value	
uint16_t	16-bit CRC value (0-0xFFFF)

Example:

```
/* get SPI0 CRC send value */
uint16_t crc_val;
crc_val = spi_crc_get(SPI0, SPI_CRC_TX);
```

### spi\_ti\_mode\_enable

The description of spi\_ti\_mode\_enable is shown as below:

Table 3-520. Function spi\_ti\_mode\_enable

Function name	spi_ti_mode_enable
Function prototype	void spi_ti_mode_enable(uint32_t spi_periph);
Function descriptions	enable SPI TI mode
Precondition	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 TI mode */
spi_ti_mode_enable(SPI0);
```

### spi\_ti\_mode\_disable

The description of spi\_ti\_mode\_disable is shown as below:

**Table 3-521. Function spi\_ti\_mode\_disable**

<b>Function name</b>	spi_ti_mode_disable
<b>Function prototype</b>	void spi_ti_mode_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable SPI TI mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 TI mode */
```

```
spi_ti_mode_disable(SPI0);
```

### spi\_nssp\_mode\_enable

The description of spi\_nssp\_mode\_enable is shown as below:

**Table 3-522. Function spi\_nssp\_mode\_enable**

<b>Function name</b>	spi_nssp_mode_enable
<b>Function prototype</b>	void spi_nssp_mode_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI NSS pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 NSS pulse mode */
spi_nssp_mode_enable(SPI0);
```

### spi\_nssp\_mode\_disable

The description of spi\_nssp\_mode\_disable is shown as below:

**Table 3-523. Function spi\_nssp\_mode\_disable**

<b>Function name</b>	spi_nssp_mode_disable
<b>Function prototype</b>	void spi_nssp_mode_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable SPI NSS pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	



<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 NSS pulse mode */
```

```
spi_nssp_mode_disable(SPI0);
```

### qspi\_enable

The description of qspi\_enable is shown as below:

**Table 3-524. Function qspi\_enable**

<b>Function name</b>	qspi_enable
<b>Function prototype</b>	void qspi_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable quad wire SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 quad wire mode */
```

```
qspi_enable(SPI0);
```

## qspi\_disable

The description of qspi\_disable is shown as below:

**Table 3-525. Function qspi\_disable**

<b>Function name</b>	qspi_disable
<b>Function prototype</b>	qspi_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable quad wire SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 quad wire mode */
```

```
qspi_disable(SPI0);
```

## qspi\_write\_enable

The description of qspi\_write\_enable is shown as below:

**Table 3-526. Function qspi\_write\_enable**

<b>Function name</b>	qspi_write_enable
<b>Function prototype</b>	void qspi_write_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable quad wire SPI write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral

<i>SPIx</i>	<i>x</i> =0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 quad wire write */
```

```
qspi_write_enable(SPI0);
```

### qspi\_read\_enable

The description of qspi\_read\_enable is shown as below:

**Table 3-527. Function qspi\_read\_enable**

Function name	qspi_read_enable
Function prototype	void qspi_read_enable(uint32_t spi_periph);
Function descriptions	enable quad wire SPI read
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	<i>x</i> =0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 quad wire read */
```

```
qspi_read_enable(SPI0);
```

### qspi\_io23\_output\_enable

The description of qspi\_io23\_output\_enable is shown as below:

Table 3-528. Function `qspi_io23_output_enable`

Function name	<code>qspi_io23_output_enable</code>
Function prototype	<code>void qspi_io23_output_enable(uint32_t spi_periph);</code>
Function descriptions	enable SPI_IO2 and SPI_IO3 pin output
Precondition	-
The called functions	-
Input parameter{in}	
<code>spi_periph</code>	SPI peripheral
<code>SPIx</code>	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 SPI_IO2 and SPI_IO3 pin output */
```

```
qspi_io23_output_enable(SPI0);
```

### **`qspi_io23_output_disable`**

The description of `qspi_io23_output_disable` is shown as below:

Table 3-529. Function `qspi_io23_output_disable`

Function name	<code>qspi_io23_output_disable</code>
Function prototype	<code>void qspi_io23_output_disable(uint32_t spi_periph);</code>
Function descriptions	disable SPI_IO2 and SPI_IO3 pin output
Precondition	-
The called functions	-
Input parameter{in}	
<code>spi_periph</code>	SPI peripheral
<code>SPIx</code>	x=0
Output parameter{out}	

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

Example:

```
/* disable SPI0 SPI_IO2 and SPI_IO3 pin output */
```

```
qspi_io23_output_disable(SPI0);
```

### spi\_i2s\_interrupt\_enable

The description of spi\_i2s\_interrupt\_enable is shown as below:

**Table 3-530. Function spi\_i2s\_interrupt\_enable**

<b>Function name</b>	spi_i2s_interrupt_enable
<b>Function prototype</b>	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	enable SPI and I2S interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt
<i>SPI_I2S_INT_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

### spi\_i2s\_interrupt\_disable

The description of spi\_i2s\_interrupt\_disable is shown as below:

**Table 3-531. Function spi\_i2s\_interrupt\_disable**

<b>Function name</b>	spi_i2s_interrupt_disable
<b>Function prototype</b>	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	disable SPI and I2S interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt
<i>SPI_I2S_INT_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

### spi\_i2s\_interrupt\_flag\_get

The description of spi\_i2s\_interrupt\_flag\_get is shown as below:

Table 3-532. Function spi\_i2s\_interrupt\_flag\_get

<b>Function name</b>	spi_i2s_interrupt_flag_get
<b>Function prototype</b>	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	get SPI and I2S interrupt status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt flag status
<i>SPI_I2S_INT_FLAG_T BE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_FLAG_R BNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_FLAG_R XORERR</i>	overrun interrupt
<i>SPI_INT_FLAG_CONF ERR</i>	config error interrupt
<i>SPI_INT_FLAG_CRCE RR</i>	CRC error interrupt
<i>I2S_INT_FLAG_TXUR ERR</i>	underrun error interrupt
<i>SPI_I2S_INT_FLAG_F ERR</i>	format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty interrupt status */
```

```

if(RESET != spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE)){

    while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));

    spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);

}

```

### spi\_i2s\_flag\_get

The description of spi\_i2s\_flag\_get is shown as below:

**Table 3-533. Function spi\_i2s\_flag\_get**

Function name	spi_i2s_flag_get
Function prototype	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);
Function descriptions	get SPI and I2S flag status
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Input parameter{in}	
flag	SPI/I2S flag status
SPI_FLAG_TBE	transmit buffer empty flag
SPI_FLAG_RBNE	receive buffer not empty flag
SPI_FLAG_TRANS	transmit on-going flag
SPI_FLAG_RXORERR	receive overrun error flag
SPI_FLAG_CONFERR	mode config error flag
SPI_FLAG_CRCERR	CRC error flag
SPI_FLAG_FERR	format error flag
I2S_FLAG_TBE	transmit buffer empty flag
I2S_FLAG_RBNE	receive buffer not empty flag
I2S_FLAG_TRANS	transmit on-going flag
I2S_FLAG_RXORERR	overrun error flag



<i>I2S_FLAG_TXURERR</i>	underrun error flag
<i>I2S_FLAG_CH</i>	channel side flag
<i>I2S_FLAG_FERR</i>	format error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty flag status */
while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

### **spi\_crc\_error\_clear**

The description of spi\_crc\_error\_clear is shown as below:

**Table 3-534. Function spi\_crc\_error\_clear**

<b>Function name</b>	spi_crc_error_clear
<b>Function prototype</b>	void spi_crc_error_clear(uint32_t spi_periph);
<b>Function descriptions</b>	clear SPI CRC error flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear SPI0 CRC error flag status */
```

```
spi_crc_error_clear(SPI0);
```

## 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, 7), general level0 timer (TIMERx, x=2, 3), general level1 timer (TIMERx, x=8, 11), general level2 timer (TIMERx, x=9, 10, 12, 13), Basic timer (TIMERx, x=5, 6). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.22.1](#), 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 3-535. 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

Registers	Descriptions
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 3-536. TIMERx firmware function**

Function name	Function description
timer_deinit	deinit a timer
timer_struct_para_init	initialize the parameters of TIMER init parameter struct with the default values
timer_init	initialize TIMER counter
timer_enable	enable a timer
timer_disable	disable a timer
timer_auto_reload_shadow_enable	enable the auto reload shadow function
timer_auto_reload_shadow_disable	disable the auto reload shadow function
timer_update_event_enable	enable the update event
timer_update_event_disable	disable the update event
timer_counter_alignment	set TIMER counter alignment mode
timer_counter_up_direction	set TIMER counter up direction
timer_counter_down_direction	set TIMER counter down direction
timer_prescaler_config	configure TIMER prescaler
timer_repetition_value_config	configure TIMER repetition register value
timer_autoreload_value_config	configure TIMER autoreload register value
timer_counter_value_config	configure TIMER counter register value
timer_counter_read	read TIMER counter value
timer_prescaler_read	read TIMER prescaler value

Function name	Function description
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

Function name	Function description
timer_channel_output_polarity_config	configure TIMER channel output polarity
timer_channel_complementary_output_polarity_config	configure TIMER channel complementary output polarity
timer_channel_output_state_config	configure TIMER channel enable state
timer_channel_complementary_output_state_config	configure TIMER channel complementary output enable state
timer_channel_input_struct_para_init	initialize the parameters of TIMER channel input parameter struct with the default values
timer_input_capture_config	configure TIMER input capture parameter
timer_channel_input_capture_prescaler_config	configure TIMER channel input capture prescaler value
timer_channel_capture_value_register_read	read TIMER channel capture compare register value
timer_input_pwm_capture_config	configure TIMER input pwm capture function
timer_hall_mode_config	configure TIMER hall sensor mode
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

Function name	Function description
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_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
timer_flag_get	get TIMER flags
timer_flag_clear	clear TIMER flags

### Structure timer\_parameter\_struct

Table 3-537. Structure timer\_parameter\_struct

Member name	Function description
prescaler	prescaler value (0~65535)
alignedmode	aligned mode (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)
counterdirection	counter direction (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)
period	period value (0~65535)
clockdivision	clock division value (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)
repetitioncounter	the counter repetition value (0~255)

### Structure timer\_break\_parameter\_struct

Table 3-538. Structure timer\_break\_parameter\_struct

Member name	Function description
runoffstate	run mode off-state (TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE)
ideloffstate	idle mode off-state (TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE)

Member name	Function description
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 3-539. Structure timer\_oc\_parameter\_struct**

Member name	Function description
outputstate	channel output state (TIMER_CCX_ENABLE, TIMER_CCX_DISABLE)
outputnstate	channel complementary output state (TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE)
ocpolarity	channel output polarity (TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW)
ocnpolarity	channel complementary output polarity (TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW)
ocidlestate	idle state of channel output (TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH)
ocnidlestate	idle state of channel complementary output (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH)

### Structure timer\_ic\_parameter\_struct

**Table 3-540. 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)

Member name	Function description
icfilter	channel input capture filter control (0~15)

## timer\_deinit

The description of timer\_deinit is shown as below:

**Table 3-541. 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..13)</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 3-542. Function timer\_struct\_para\_init**

<b>Function name</b>	timer_struct_para_init
<b>Function prototype</b>	void timer_struct_para_init(timer_parameter_struct* initpara);
<b>Function descriptions</b>	initialize the parameters of TIMER init parameter struct with the default values
<b>Precondition</b>	-



<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>initpara</b>	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-537. Structure timer parameter struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize TIMER init parameter struct with a default value */
```

```
timer_parameter_struct timer_initpara;
```

```
timer_struct_para_init(timer_initpara);
```

### timer\_init

The description of timer\_init is shown as below:

**Table 3-543. Function timer\_init**

<b>Function name</b>	timer_init
<b>Function prototype</b>	void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);
<b>Function descriptions</b>	initialize TIMER counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>initpara</b>	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-537. Structure timer parameter struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* initialize TIMER0 */

timer_parameter_struct timer_initpara;

timer_initpara.prescaler      = 107;

timer_initpara.alignedmode    = TIMER_COUNTER_EDGE;

timer_initpara.counterdirection = TIMER_COUNTER_UP;

timer_initpara.period         = 999;

timer_initpara.clockdivision   = TIMER_CKDIV_DIV1;

timer_initpara.repetitioncounter = 1;

timer_init(TIMER0,&timer_initpara);
```

### timer\_enable

The description of timer\_enable is shown as below:

**Table 3-544. Function timer\_enable**

<b>Function name</b>	timer_enable
<b>Function prototype</b>	void timer_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable a timer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 */
```

timer\_enable (TIMER0);

## timer\_disable

The description of timer\_disable is shown as below:

**Table 3-545. Function timer\_disable**

<b>Function name</b>	timer_disable
<b>Function prototype</b>	void timer_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable a timer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 */
```

```
timer_disable (TIMER0);
```

## timer\_auto\_reload\_shadow\_enable

The description of timer\_auto\_reload\_shadow\_enable is shown as below:

**Table 3-546. Function timer\_auto\_reload\_shadow\_enable**

<b>Function name</b>	timer_auto_reload_shadow_enable
<b>Function prototype</b>	void timer_auto_reload_shadow_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable the auto reload shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable (TIMER0);
```

### timer\_auto\_reload\_shadow\_disable

The description of timer\_auto\_reload\_shadow\_disable is shown as below:

**Table 3-547. Function timer\_auto\_reload\_shadow\_disable**

<b>Function name</b>	timer_auto_reload_shadow_disable
<b>Function prototype</b>	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable the auto reload shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable (TIMER0);
```

## timer\_update\_event\_enable

The description of timer\_update\_event\_enable is shown as below:

**Table 3-548. Function timer\_update\_event\_enable**

<b>Function name</b>	timer_update_event_enable
<b>Function prototype</b>	void timer_update_event_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable the update event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 the update event */
timer_update_event_enable (TIMER0);
```

## timer\_update\_event\_disable

The description of timer\_update\_event\_disable is shown as below:

**Table 3-549. 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</i> ( <i>x</i> =0..13)	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 3-550. 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</i> ( <i>x</i> =0..4,7)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>aligned</b>	alignment mode
<i>TIMER_COUNTER_EDGE</i>	No center-aligned mode (edge-aligned mode). The direction of the counter is specified by the DIR bit.
<i>TIMER_COUNTER_COUNTER_DOWN</i>	Center-aligned and counting down assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in <i>TIMERx_CHCTL0</i> register). Only when the counter is counting down, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_COUNTER_UP</i>	Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in <i>TIMERx_CHCTL0</i> register). Only when the counter is counting up, compare

	interrupt flag of channels can be set.
<i>TIMER_COUNTER_CEN TER_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 3-551. Function timer\_counter\_up\_direction**

<b>Function name</b>	timer_counter_up_direction
<b>Function prototype</b>	void timer_counter_up_direction(uint32_t timer_periph);
<b>Function descriptions</b>	set TIMER counter up direction
<b>Precondition</b>	set TIMER counter no center-aligned mode (edge-aligned mode)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</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 3-552. Function timer\_counter\_down\_direction**

<b>Function name</b>	timer_counter_down_direction
<b>Function prototype</b>	void timer_counter_down_direction(uint32_t timer_periph);
<b>Function descriptions</b>	set TIMER counter down direction
<b>Precondition</b>	set TIMER counter no center-aligned mode (edge-aligned mode)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</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 3-553. 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, uint8_t pscreload);
<b>Function descriptions</b>	configure TIMER prescaler
<b>Precondition</b>	-
<b>The called functions</b>	-



Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Input parameter{in}	
<b>prescaler</b>	prescaler value (0~65535)
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 prescaler */
```

```
timer_prescaler_config (TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

### timer\_repetition\_value\_config

The description of timer\_repetition\_value\_config is shown as below:

**Table 3-554. Function timer\_repetition\_value\_config**

<b>Function name</b>	timer_repetition_value_config
<b>Function prototype</b>	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
<b>Function descriptions</b>	configure TIMER repetition register value
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>repetition</b>	the counter repetition value (0~255)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 repetition register value */
```

```
timer_repetition_value_config (TIMER0, 98);
```

### timer\_autoreload\_value\_config

The description of timer\_autoreload\_value\_config is shown as below:

**Table 3-555. Function timer\_autoreload\_value\_config**

<b>Function name</b>	timer_autoreload_value_config
<b>Function prototype</b>	void timer_autoreload_value_config(uint32_t timer_periph, uint16_t autoreload);
<b>Function descriptions</b>	configure TIMER autoreload register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>autoreload</b>	the counter auto-reload value (0-0xFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-556. 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, uint16_t counter);
<b>Function descriptions</b>	configure TIMER counter register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>counter</b>	the counter value (0-0xFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 counter register value */
timer_counter_value_config (TIMER0);
```

### timer\_counter\_read

The description of timer\_counter\_read is shown as below:

**Table 3-557. Function timer\_counter\_read**

<b>Function name</b>	timer_counter_read
<b>Function prototype</b>	uint32_t timer_counter_read(uint32_t timer_periph);

<b>Function descriptions</b>	read TIMER counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	counter value(0x0000~0xFFFF)

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 3-558. Function timer\_prescaler\_read**

<b>Function name</b>	timer_prescaler_read
<b>Function prototype</b>	uint16_t timer_prescaler_read(uint32_t timer_periph);
<b>Function descriptions</b>	read TIMER prescaler value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

<b>uint16_t</b>	prescaler register value (0x0000~0xFFFF)
-----------------	--

Example:

```
/* read TIMER0 prescaler value */
```

```
uint16_t i = 0;
```

```
i = timer_prescaler_read (TIMER0);
```

### timer\_single\_pulse\_mode\_config

The description of timer\_single\_pulse\_mode\_config is shown as below:

**Table 3-559. Function timer\_single\_pulse\_mode\_config**

<b>Function name</b>	timer_single_pulse_mode_config
<b>Function prototype</b>	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
<b>Function descriptions</b>	configure TIMER single pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..8,11)</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 3-560. 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(x=0..13)</i>	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: <ul style="list-style-type: none"> <li>- The UPG bit is set</li> <li>- The counter generates an overflow or underflow event</li> <li>- The slave mode controller generates an update event</li> </ul>
<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 3-561. Function timer\_dma\_enable

Function name	timer_dma_enable
Function prototype	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
Function descriptions	enable the TIMER DMA
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMEx	please refer to the following parameters
Input parameter{in}	
dma	timer DMA source enable
TIMER_DMA_UPD	update DMA enable, TIMEx(x=0..7)
TIMER_DMA_CH0D	channel 0 DMA enable, TIMEx(x=0..4,7)
TIMER_DMA_CH1D	channel 1 DMA enable, TIMEx(x=0..4,7)
TIMER_DMA_CH2D	channel 2 DMA enable, TIMEx(x=0..4,7)
TIMER_DMA_CH3D	channel 3 DMA enable, TIMEx(x=0..4,7)
TIMER_DMA_CMTD	commutation DMA request enable, TIMEx(x=0,7)
TIMER_DMA_TRGD	trigger DMA enable, TIMEx(x=0..4,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TIMER0 update DMA */
```

```
timer_dma_enable (TIMER0, TIMER_DMA_UPD);
```

### timer\_dma\_disable

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

Table 3-562. Function timer\_dma\_disable

Function name	timer_dma_disable
Function prototype	void timer_dma_disable (uint32_t timer_periph, uint16_t dma);
Function descriptions	disable the TIMER DMA
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMEx	please refer to the following parameters
Input parameter{in}	
dma	timer DMA source disable
TIMER_DMA_UPD	update DMA disable, TIMEx(x=0..7)
TIMER_DMA_CH0D	channel 0 DMA disable, TIMEx(x=0..4,7)
TIMER_DMA_CH1D	channel 1 DMA disable, TIMEx(x=0..4,7)
TIMER_DMA_CH2D	channel 2 DMA disable, TIMEx(x=0..4,7)
TIMER_DMA_CH3D	channel 3 DMA disable, TIMEx(x=0..4,7)
TIMER_DMA_CMTD	commutation DMA request disable, TIMEx(x=0,7)
TIMER_DMA_TRGD	trigger DMA disable, TIMEx(x=0..4,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 update DMA */
```

```
timer_dma_disable (TIMER0, TIMER_DMA_UPD);
```

### timer\_channel\_dma\_request\_source\_select

The description of timer\_channel\_dma\_request\_source\_select is shown as below:



Table 3-563. Function timer\_channel\_dma\_request\_source\_select

<b>Function name</b>	timer_channel_dma_request_source_select
<b>Function prototype</b>	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
<b>Function descriptions</b>	channel DMA request source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>dma_request</b>	channel DMA request source selection
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	DMA request of channel n is sent when channel y event occurs
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	DMA request of channel n is sent when update event occurs
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* TIMER0 channel DMA request of channel n is sent when channel y event occurs */
```

```
timer_channel_dma_request_source_select(TIMER0,
TIMER_DMAREQUEST_CHANNELEVENT);
```

### timer\_dma\_transfer\_config

The description of timer\_dma\_transfer\_config is shown as below:

Table 3-564. Function timer\_dma\_transfer\_config

<b>Function name</b>	timer_dma_transfer_config
<b>Function prototype</b>	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);

<b>Function descriptions</b>	configure the TIMER DMA transfer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma_baseaddr</b>	DMA transfer access start address
<i>TIMER_DMACFG_DMA TA_CTL0</i>	DMA transfer address is TIMER_CTL0, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CTL1</i>	DMA transfer address is TIMER_CTL1, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_SMCFG</i>	DMA transfer address is TIMER_SMCFG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_DMAINTEN</i>	DMA transfer address is TIMER_DMAINTEN, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_INTF</i>	DMA transfer address is TIMER_INTF, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_SWEVG</i>	DMA transfer address is TIMER_SWEVG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CHCTL0</i>	DMA transfer address is TIMER_CHCTL0, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CHCTL1</i>	DMA transfer address is TIMER_CHCTL1, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CHCTL2</i>	DMA transfer address is TIMER_CHCTL2, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CNT</i>	DMA transfer address is TIMER_CNT, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_PSC</i>	DMA transfer address is TIMER_PSC, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CAR</i>	MA transfer address is TIMER_CAR, TIMERx(x=0..4,7)

<i>TIMER_DMACFG_DMA TA_CREP</i>	DMA transfer address is TIMER_CREP, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMA TA_CH0CV</i>	DMA transfer address is TIMER_CH0CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CH1CV</i>	DMA transfer address is TIMER_CH1CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CH2CV</i>	DMA transfer address is TIMER_CH2CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CH3CV</i>	DMA transfer address is TIMER_CH3CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CCHP</i>	DMA transfer address is TIMER_CCHP, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMA TA_DMACFG</i>	DMA transfer address is TIMER_DMACFG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_DMATB</i>	DMA transfer address is TIMER_DMATB, TIMERx(x=0..4,7)
<b>Input parameter{in}</b>	
<b>dma_lenth</b>	DMA transfer count
<i>TIMER_DMACFG_DMA TC_xTRANSFER</i>	x=1..18, DMA transfer x time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the TIMER0 DMA transfer */
```

```
timer_dma_transfer_config(TIMER0, TIMER_DMACFG_DMATA_CTL0,  
TIMER_DMACFG_DMATC_5TRANSFER);
```

### **timer\_event\_software\_generate**

The description of timer\_event\_software\_generate is shown as below:

Table 3-565. 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_UPDATE</i>	update event, TIMERx(x=0..13)
<i>TIMER_EVENT_SRC_C0G</i>	channel 0 capture or compare event generation, TIMERx(x=0..4,7..13)
<i>TIMER_EVENT_SRC_C1G</i>	channel 1 capture or compare event generation, TIMERx(x=0..4,7,8,11)
<i>TIMER_EVENT_SRC_C2G</i>	channel 2 capture or compare event generation, TIMERx(x=0..4,7)
<i>TIMER_EVENT_SRC_C3G</i>	channel 3 capture or compare event generation, TIMERx(x=0..4,7)
<i>TIMER_EVENT_SRC_CMTG</i>	channel commutation event generation, TIMERx(x=0,7)
<i>TIMER_EVENT_SRC_TRIGGER</i>	trigger event generation, TIMERx(x=0..4,7,8,11)
<i>TIMER_EVENT_SRC_BREAK</i>	break event generation, TIMERx(x=0,7)
<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 3-566. Function timer\_break\_struct\_para\_init**

Function name	timer_break_struct_para_init
Function prototype	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
Function descriptions	initialize the parameters of TIMER break parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
breakpara	TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-538. Structure timer break parameter struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER break parameter struct with a default value */
```

```
timer_break_parameter_struct timer_breakpara;
```

```
timer_break_struct_para_init(timer_breakpara);
```

### timer\_break\_config

The description of timer\_break\_config is shown as below:

**Table 3-567. Function timer\_break\_config**

Function name	timer_break_config
Function prototype	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
<b>TIMER<sub>x</sub>(x=0,7)</b>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>breakpara</b>	TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-538. 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 3-568. Function timer\_break\_enable**

<b>Function name</b>	timer_break_enable
----------------------	--------------------

<b>Function prototype</b>	void timer_break_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable TIMER break function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 break function*/
```

```
timer_break_enable (TIMER0);
```

### timer\_break\_disable

The description of timer\_break\_disable is shown as below:

**Table 3-569. Function timer\_break\_disable**

<b>Function name</b>	timer_break_disable
<b>Function prototype</b>	void timer_break_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER break function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* disable TIMER0 break function*/
```

```
timer_break_disable (TIMER0);
```

### timer\_automatic\_output\_enable

The description of timer\_automatic\_output\_enable is shown as below:

**Table 3-570. Function timer\_automatic\_output\_enable**

Function name	timer_automatic_output_enable
Function prototype	void timer_automatic_output_enable(uint32_t timer_periph);
Function descriptions	enable TIMER output automatic function
Precondition	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0,7)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER0 output automatic function */
```

```
timer_automatic_output_enable (TIMER0);
```

### timer\_automatic\_output\_disable

The description of timer\_automatic\_output\_disable is shown as below:

**Table 3-571. Function timer\_automatic\_output\_disable**

Function name	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=0,7)</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 3-572. 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=0,7)</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 3-573. 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=0,7)</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 3-574. Function timer\_channel\_control\_shadow\_update\_config**

<b>Function name</b>	timer_channel_control_shadow_update_config
<b>Function prototype</b>	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint8_t ccuctl);
<b>Function descriptions</b>	configure commutation control shadow register update control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-575. Function timer\_channel\_output\_struct\_para\_init**

<b>Function name</b>	timer_channel_output_struct_para_init
<b>Function prototype</b>	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpa);
<b>Function descriptions</b>	initialize the parameters of TIMER channel output parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ocpara</b>	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-539. Structure timer oc parameter struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize TIMER channel output parameter struct with a default value */
timer_oc_parameter_struct timer_ocinitpara;
timer_channel_output_struct_para_init(timer_ocinitpara);
```

## timer\_channel\_output\_config

The description of timer\_channel\_output\_config is shown as below:

**Table 3-576. Function timer\_channel\_output\_config**

<b>Function name</b>	timer_channel_output_config
<b>Function prototype</b>	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpa);
<b>Function descriptions</b>	configure TIMER channel output function
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>ocpara</b>	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-539. Structure timer oc parameter struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure TIMER0 channel 0 output function */

timer_oc_parameter_struct timer_ocinitpara;

timer_ocinitpara.outputstate = TIMER_CCX_ENABLE;

timer_ocinitpara.outputnstate = TIMER_CCXN_ENABLE;

timer_ocinitpara.ocpolarity = TIMER_OC_POLARITY_HIGH;

timer_ocinitpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;

timer_ocinitpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;

timer_ocinitpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;

timer_channel_output_config(TIMER0, TIMER_CH_0, &timer_ocinitpara);

```

## timer\_channel\_output\_mode\_config

The description of timer\_channel\_output\_mode\_config is shown as below:

**Table 3-577. Function timer\_channel\_output\_mode\_config**

<b>Function name</b>	timer_channel_output_mode_config
<b>Function prototype</b>	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);
<b>Function descriptions</b>	configure TIMER channel output compare mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<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_HIG H</i>	force high mode
<i>TIMER_OC_MODE_PW M0</i>	PWM mode 0
<i>TIMER_OC_MODE_PW M1</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 3-578. Function timer\_channel\_output\_pulse\_value\_config**

<b>Function name</b>	timer_channel_output_pulse_value_config
<b>Function prototype</b>	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);
<b>Function descriptions</b>	configure TIMER channel output pulse value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))

<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>pulse</b>	channel output pulse value (0~65535)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-579. Function timer\_channel\_output\_shadow\_config**

<b>Function name</b>	timer_channel_output_shadow_config
<b>Function prototype</b>	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
<b>Function descriptions</b>	configure TIMER channel output shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))



<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>ocshadow</b>	channel output shadow state
<i>TIMER_OC_SHADOW_ENABLE</i>	channel output shadow state enable
<i>TIMER_OC_SHADOW_DISABLE</i>	channel output shadow state disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*configure TIMER0 channel 0 output shadow function */
timer_channel_output_shadow_config (TIMER0, TIMER_CH_0,
TIMER_OC_SHADOW_ENABLE);
```

### timer\_channel\_output\_fast\_config

The description of timer\_channel\_output\_fast\_config is shown as below:

**Table 3-580. Function timer\_channel\_output\_fast\_config**

<b>Function name</b>	timer_channel_output_fast_config
<b>Function prototype</b>	void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
<b>Function descriptions</b>	configure TIMER channel output fast function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured

<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>ocfast</b>	channel output fast function
<i>TIMER_OC_FAST_ENA BLE</i>	channel output fast function enable
<i>TIMER_OC_FAST_DIS ABLE</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 3-581. 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(x=0..4,7)</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
<i>TIMER_CH_1</i>	TIMER channel 1
<i>TIMER_CH_2</i>	TIMER channel 2
<i>TIMER_CH_3</i>	TIMER channel 3
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output clear function */
timer_channel_output_clear_config (TIMER0, TIMER_CH_0,
TIMER_OC_CLEAR_ENABLE);
```

### timer\_channel\_output\_polarity\_config

The description of timer\_channel\_output\_polarity\_config is shown as below:

**Table 3-582. Function timer\_channel\_output\_polarity\_config**

<b>Function name</b>	timer_channel_output_polarity_config
<b>Function prototype</b>	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
<b>Function descriptions</b>	configure TIMER channel output polarity
<b>Precondition</b>	-
<b>The called functions</b>	-

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..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output polarity */
```

```
timer_channel_output_polarity_config (TIMER0, TIMER_CH_0,  
TIMER_OC_POLARITY_HIGH);
```

### timer\_channel\_complementary\_output\_polarity\_config

The description of timer\_channel\_complementary\_output\_polarity\_config is shown as below:

**Table 3-583. 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 ocpolarity);

<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(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<i>TIMER_CH_2</i>	TIMER channel 2
<b>Input parameter{in}</b>	
<b>ocpolarity</b>	channel complementary output polarity
<i>TIMER_OCN_POLARITY_HIGH</i>	channel complementary output polarity is high
<i>TIMER_OCN_POLARITY_LOW</i>	channel complementary output polarity is low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config (TIMER0, TIMER_CH_0,  
TIMER_OCN_POLARITY_HIGH);
```

### timer\_channel\_output\_state\_config

The description of timer\_channel\_output\_state\_config is shown as below:

**Table 3-584. Function timer\_channel\_output\_state\_config**

<b>Function name</b>	timer_channel_output_state_config
----------------------	-----------------------------------

<b>Function prototype</b>	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
<b>Function descriptions</b>	configure TIMER channel enable state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<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 3-585. 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(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<i>TIMER_CH_2</i>	TIMER channel 2
<b>Input parameter{in}</b>	
<b>state</b>	TIMER channel complementary output enable state
<i>TIMER_CCXN_ENABLE</i>	channel complementary enable
<i>TIMER_CCXN_DISABLE</i>	channel complementary disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output enable state */
```

```
timer_channel_complementary_output_state_config (TIMER0, TIMER_CH_0,
TIMER_CCXN_ENABLE);
```

## timer\_channel\_input\_struct\_para\_init

The description of timer\_channel\_input\_struct\_para\_init is shown as below:

**Table 3-586. 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 3-540. 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 3-587. Function timer\_input\_capture\_config**

<b>Function name</b>	timer_input_capture_config
<b>Function prototype</b>	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
<b>Function descriptions</b>	configure TIMER input capture parameter
<b>Precondition</b>	-



<b>The called functions</b>	timer_channel_input_capture_prescaler_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>icpara</b>	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-540. Structure timer ic parameter struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure TIMER0 input capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter      = 0x0;

timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);

```

### timer\_channel\_input\_capture\_prescaler\_config

The description of timer\_channel\_input\_capture\_prescaler\_config is shown as below:

Table 3-588. Function timer\_channel\_input\_capture\_prescaler\_config

<b>Function name</b>	timer_channel_input_capture_prescaler_config
<b>Function prototype</b>	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);
<b>Function descriptions</b>	configure TIMER channel input capture prescaler value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-589. Function timer\_channel\_capture\_value\_register\_read**

<b>Function name</b>	timer_channel_capture_value_register_read
<b>Function prototype</b>	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
<b>Function descriptions</b>	read TIMER channel capture compare register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	channel capture compare register value (0x0000~0xFFFF)

Example:

```
/* read TIMER0 channel 0 capture compare register value */
```

```
uint32_t ch0_value = 0;
```

```
ch0_value = timer_channel_capture_value_register_read (TIMER0, TIMER_CH_0);
```

## timer\_input\_pwm\_capture\_config

The description of timer\_input\_pwm\_capture\_config is shown as below:

**Table 3-590. Function timer\_input\_pwm\_capture\_config**

<b>Function name</b>	timer_input_pwm_capture_config
<b>Function prototype</b>	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
<b>Function descriptions</b>	configure TIMER input pwm capture function
<b>Precondition</b>	-
<b>The called functions</b>	timer_channel_input_capture_prescaler_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<b>Input parameter{in}</b>	
<b>icpwm</b>	TIMER channel input pwm parameter struct, the structure members can refer to <a href="#">Table 3-540. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure TIMER0 input pwm capture parameter */
timer_ic_parameter_struct timer_initpara;

timer_initpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_initpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_initpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_initpara.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 3-591. Function timer\_hall\_mode\_config**

<b>Function name</b>	timer_hall_mode_config
<b>Function prototype</b>	void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode);
<b>Function descriptions</b>	configure TIMER hall sensor mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>hallmode</b>	TIMER hall sensor mode state
<i>TIMER_HALLINTERFACE_ENABLE</i>	TIMER hall sensor mode enable
<i>TIMER_HALLINTERFACE_DISABLE</i>	TIMER hall sensor mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 hall sensor mode */
timer_hall_mode_config (TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

### timer\_input\_trigger\_source\_select

The description of timer\_input\_trigger\_source\_select is shown as below:

**Table 3-592. Function timer\_input\_trigger\_source\_select**

<b>Function name</b>	timer_input_trigger_source_select
----------------------	-----------------------------------

<b>Function prototype</b>	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);
<b>Function descriptions</b>	select TIMER input trigger source
<b>Precondition</b>	SMC[2:0] = 000
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI0</i>	Internal trigger input 0 (ITI0, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI1</i>	Internal trigger input 0 (ITI1, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI2</i>	Internal trigger input 0 (ITI2, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI3</i>	Internal trigger input 0 (ITI3, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CIO edge flag (CIOF_ED, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output (CIOFE0, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_C1FE1</i>	channel 1 input Filtered output (C1FE1, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_ETIFP</i>	External trigger input filter output(ETIFP, TIMERx(x=0..4,7) )
<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 3-593. Function timer\_master\_output\_trigger\_source\_select**

Function name	timer_master_output_trigger_source_select
Function prototype	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
Function descriptions	select TIMER master mode output trigger source
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..7)	TIMER peripheral selection
Input parameter{in}	
outrigger	master mode control
TIMER_TRI_OUT_SRC_RESET	Reset. When the UPG bit in the TIMERx_SWEVG register is set or a reset is generated by the slave mode controller, a TRGO pulse occurs. And in the latter case, the signal on TRGO is delayed compared to the actual reset
TIMER_TRI_OUT_SRC_ENABLE	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.
TIMER_TRI_OUT_SRC_UPDATE	Update. In this mode the master mode controller selects the update event as TRGO.
TIMER_TRI_OUT_SRC_CC0	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.
TIMER_TRI_OUT_SRC_O0CPRE	Compare. In this mode the master mode controller selects the O0CPRE signal is used as TRGO.

<i>TIMER_TRI_OUT_SRC_O1CPRE</i>	Compare. In this mode the master mode controller selects the O1CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O2CPRE</i>	Compare. In this mode the master mode controller selects the O2CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O3CPRE</i>	Compare. In this mode the master mode controller selects the O3CPRE signal is used as TRGO.
<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 3-594. Function timer\_slave\_mode\_select**

<b>Function name</b>	timer_slave_mode_select
<b>Function prototype</b>	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
<b>Function descriptions</b>	select TIMER slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>slavemode</b>	slave mode
<i>TIMER_SLAVE_MODE_DISABLE</i>	slave mode disable(x=0..4,7,8,11)
<i>TIMER_ENCODER_MODE0</i>	encoder mode 0(x=0..4,7)



<i>TIMER_ENCODER_MODE1</i>	encoder mode 1(x=0..4,7)
<i>TIMER_ENCODER_MODE2</i>	encoder mode 2(x=0..4,7)
<i>TIMER_SLAVE_MODE_RESTART</i>	restart mode(x=0..4,7,8,11)
<i>TIMER_SLAVE_MODE_PAUSE</i>	pause mode(x=0..4,7,8,11)
<i>TIMER_SLAVE_MODE_EVENT</i>	event mode(x=0..4,7,8,11)
<i>TIMER_SLAVE_MODE_EXTERNAL0</i>	external clock mode 0(x=0..4,7,8,11)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER0 slave mode */
```

```
timer_slave_mode_select (TIMER0, TIMER_ENCODER_MODE0);
```

### timer\_master\_slave\_mode\_config

The description of timer\_master\_slave\_mode\_config is shown as below:

**Table 3-595. Function timer\_master\_slave\_mode\_config**

<b>Function name</b>	timer_master_slave_mode_config
<b>Function prototype</b>	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);
<b>Function descriptions</b>	configure TIMER master slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx(x=0..4,7,8,11)</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 3-596. Function timer\_external\_trigger\_config**

<b>Function name</b>	timer_external_trigger_config
<b>Function prototype</b>	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER external trigger input
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	external trigger prescaler
<i>TIMER_EXT_TRIG_PSC_0</i>	no divided

OFF	
TIMER_EXT_TRI_PSC_DIV2	divided by 2
TIMER_EXT_TRI_PSC_DIV4	divided by 4
TIMER_EXT_TRI_PSC_DIV8	divided by 8
<b>Input parameter{in}</b>	
expolarity	external trigger polarity
TIMER_ETP_FALLING	active low or falling edge active
TIMER_ETP_RISING	active high or rising edge active
<b>Input parameter{in}</b>	
extfilter	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 3-597. 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>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
Input parameter{in}	
<b>decomode</b>	quadrature decoder mode
<i>TIMER_ENCODER_MODE0</i>	counter counts on CI0FE0 edge depending on CI1FE1 level
<i>TIMER_ENCODER_MODE1</i>	counter counts on CI1FE1 edge depending on CI0FE0 level
<i>TIMER_ENCODER_MODE2</i>	counter counts on both CI0FE0 and CI1FE1 edges depending on the level of the other input
Input parameter{in}	
<b>ic0polarity</b>	IC0 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
Input parameter{in}	
<b>ic1polarity</b>	IC1 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config (TIMER0, TIMER_ENCODER_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 3-598. Function timer\_internal\_clock\_config**

<b>Function name</b>	timer_internal_clock_config
<b>Function prototype</b>	void timer_internal_clock_config(uint32_t timer_periph);
<b>Function descriptions</b>	configure TIMER internal clock mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 internal clock mode */
```

```
timer_internal_clock_config (TIMER0);
```

## timer\_internal\_trigger\_as\_external\_clock\_config

The description of timer\_internal\_trigger\_as\_external\_clock\_config is shown as below:

**Table 3-599. Function timer\_internal\_trigger\_as\_external\_clock\_config**

<b>Function name</b>	timer_internal_trigger_as_external_clock_config
<b>Function prototype</b>	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
<b>Function descriptions</b>	configure TIMER the internal trigger as external clock input
<b>Precondition</b>	-
<b>The called functions</b>	timer_input_trigger_source_select
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI0</i>	Internal trigger input 0 (ITI0)
<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 3-600. 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</i> ( <i>x</i> =0..4,7,8,11)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extrigger</b>	external trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CI0 edge flag (CIOF_ED)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output (CIOFE0)
<i>TIMER_SMCFG_TRGS</i> <i>EL_C11FE1</i>	channel 1 input Filtered output (C11FE1)
<b>Input parameter{in}</b>	
<b>expolarity</b>	external trigger polarity
<i>TIMER_IC_POLARITY_</i> <i>RISING</i>	active high or rising edge active
<i>TIMER_IC_POLARITY_</i> <i>FALLING</i>	active low or falling edge active
<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 CIOFE0 as external clock input */
timer_external_trigger_as_external_clock_config (TIMER0,
TIMER_SMCFG_TRGSEL_CIOFE0, TIMER_IC_POLARITY_RISING, 0);
```

### timer\_external\_clock\_mode0\_config

The description of timer\_external\_clock\_mode0\_config is shown as below:

**Table 3-601. 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..4,7,8,11)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
<b>Input parameter{in}</b>	
<b>expolarity</b>	ETI external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	ETI external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 the external clock 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 3-602. Function timer\_external\_clock\_mode1\_config**

<b>Function name</b>	timer_external_clock_mode1_config
<b>Function prototype</b>	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external clock mode1
<b>Precondition</b>	-
<b>The called functions</b>	timer_external_trigger_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
TIMERx(x=0..4,7)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	ETI external trigger prescaler
TIMER_EXT_TRI_PSC_OFF	no divided
TIMER_EXT_TRI_PSC_DIV2	divided by 2
TIMER_EXT_TRI_PSC_DIV4	divided by 4
TIMER_EXT_TRI_PSC_DIV8	divided by 8
<b>Input parameter{in}</b>	
<b>expolarity</b>	ETI external trigger polarity
TIMER_ETP_FALLING	active low or falling edge active
TIMER_ETP_RISING	active high or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	ETI external trigger filter control (0~15)
<b>Output parameter{out}</b>	

-	-
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 3-603. 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>TIMER<sub>x</sub>(x=0..4,7)</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 3-604. Function timer\_write\_chxval\_register\_config

<b>Function name</b>	timer_write_chxval_register_config
<b>Function prototype</b>	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
<b>Function descriptions</b>	configure TIMER write CHxVAL register selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7..13)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>ccsel</b>	write CHxVAL register selection
<i>TIMER_CHVSEL_DISABLE</i>	no effect
<i>TIMER_CHVSEL_ENABLE</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 3-605. Function timer\_output\_value\_selection\_config

<b>Function name</b>	timer_output_value_selection_config
<b>Function prototype</b>	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);
<b>Function descriptions</b>	configure TIMER output value selection

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx (x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>outsel</b>	output value selection
<i>TIMER_OUTSEL_DISABLE</i>	no effect
<i>TIMER_OUTSEL_ENABLE</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\_interrupt\_enable

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

**Table 3-606. Function timer\_interrupt\_enable**

<b>Function name</b>	timer_interrupt_enable
<b>Function prototype</b>	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable the TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters

Input parameter{in}	
<b>interrupt</b>	timer interrupt enable source
<i>TIMER_INT_UP</i>	update interrupt enable, TIMEx (x=0..13)
<i>TIMER_INT_CH0</i>	channel 0 interrupt enable, TIMEx (x=0..4,7..13)
<i>TIMER_INT_CH1</i>	channel 1 interrupt enable, TIMEx (x=0..4,7,8,11)
<i>TIMER_INT_CH2</i>	channel 2 interrupt enable, TIMEx (x=0..4,7)
<i>TIMER_INT_CH3</i>	channel 3 interrupt enable , TIMEx (x=0..4,7)
<i>TIMER_INT_CMT</i>	commutation interrupt enable, TIMEx (x=0,7)
<i>TIMER_INT_TRG</i>	trigger interrupt enable, TIMEx (x=0..4,7,8,11)
<i>TIMER_INT_BRK</i>	break interrupt enable, TIMEx (x=0,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TIMER0 update interrupt */
```

```
timer_interrupt_enable (TIMER0, TIMER_INT_UP);
```

### timer\_interrupt\_disable

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

**Table 3-607. Function timer\_interrupt\_disable**

<b>Function name</b>	timer_interrupt_disable
<b>Function prototype</b>	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable the TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMEx</i>	please refer to the following parameters

Input parameter{in}	
<b>interrupt</b>	timer interrupt disable source
<i>TIMER_INT_UP</i>	update interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..13)
<i>TIMER_INT_CH0</i>	channel 0 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4,7..13)
<i>TIMER_INT_CH1</i>	channel 1 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)
<i>TIMER_INT_CH2</i>	channel 2 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_INT_CH3</i>	channel 3 interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_INT_CMT</i>	commutation interrupt disable, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_TRG</i>	trigger interrupt disable, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)
<i>TIMER_INT_BRK</i>	break interrupt disable, <i>TIMERx</i> ( <i>x</i> =0,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 update interrupt */
timer_interrupt_disable (TIMER0, TIMER_INT_UP);
```

### timer\_interrupt\_flag\_get

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

**Table 3-608. 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 interrupt);
<b>Function descriptions</b>	get timer interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..13)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7..13)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_INT_FLAG_CMT</i>	channel commutation interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7,8,11)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TIMERO0 update interrupt flag */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = timer_interrupt_flag_get (TIMERO0, TIMER_INT_FLAG_UP);
```

### timer\_interrupt\_flag\_clear

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

**Table 3-609. 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 interrupt);
<b>Function descriptions</b>	clear TIMER interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>interrupt</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..13)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7..13)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_INT_FLAG_CMT</i>	channel commutation interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7,8,11)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TIMER0 update interrupt flag */
```

```
timer_interrupt_flag_clear (TIMER0, TIMER_INT_FLAG_UP);
```

### timer\_flag\_get

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

**Table 3-610. 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, <i>TIMERx</i> ( <i>x</i> =0..13)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7..13)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_TRG</i>	trigger flag, <i>TIMERx</i> ( <i>x</i> =0,7,8,11)
<i>TIMER_FLAG_BRK</i>	break flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..4,7..11)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<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 3-611. Function timer\_flag\_clear

Function name	timer_flag_clear
Function prototype	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
Function descriptions	clear TIMER flags
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
flag	the timer interrupt flags
TIMER_FLAG_UP	update flag, TIMERx(x=0..13)
TIMER_FLAG_CH0	channel 0 flag, TIMERx(x=0..4,7..13)
TIMER_FLAG_CH1	channel 1 flag, TIMERx(x=0..4,7,8,11)
TIMER_FLAG_CH2	channel 2 flag, TIMERx(x=0..4,7)
TIMER_FLAG_CH3	channel 3 flag, TIMERx(x=0..4,7)
TIMER_FLAG_CMT	channel commutation flag, TIMERx(x=0,7)
TIMER_FLAG_TRG	trigger flag, TIMERx(x=0,7,8,11)
TIMER_FLAG_BRK	break flag, TIMERx(x=0,7)
TIMER_FLAG_CH0O	channel 0 overcapture flag, TIMERx(x=0..4,7..11)
TIMER_FLAG_CH1O	channel 1 overcapture flag, TIMERx(x=0..4,7,8,11)
TIMER_FLAG_CH2O	channel 2 overcapture flag, TIMERx(x=0..4,7)
TIMER_FLAG_CH3O	channel 3 overcapture flag, TIMERx(x=0..4,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TIMER0 update flags */
```

timer\_flag\_clear (TIMER0, TIMER\_FLAG\_UP);

## 3.23. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.23.1](#), the USART firmware functions are introduced in chapter [3.23.2](#).

### 3.23.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

**Table 3-612. USART Registers**

Registers	Descriptions
USART_STAT0	Status register 0
USART_DATA	Data register
USART_BAUD	Baud rate register
USART_CTL0	Control register 0
USART_CTL1	Control register 1
USART_CTL2	Control register 2
USART_GP	Guard time and prescaler register
USART_CTL3	Control register 3
USART_RT	Receiver timeout register
USART_STAT1	Status register 1

### 3.23.2. Descriptions of Peripheral functions

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

**Table 3-613. USART firmware function**

Function name	Function description
usart_deinit	reset USART/UART
usart_baudrate_set	configure USART baud rate value
usart_parity_config	configure USART parity

Function name	Function description
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_receiver_timeout_enable	enable receiver timeout
usart_receiver_timeout_disable	disable receiver timeout
usart_receiver_timeout_threshold_config	configure receiver timeout threshold
usart_data_transmit	USART transmit data function
usart_data_receive	USART receive data function
usart_address_config	configure the address of the USART in wake up by address match mode
usart_mute_mode_enable	enable mute mode
usart_mute_mode_disable	disable mute mode
usart_mute_mode_wakeup_config	configure wakeup method in mute mode
usart_lin_mode_enable	enable LIN mode
usart_lin_mode_disable	disable LIN mode
usart_lin_break_dection_length_config	configure LIN break frame length
usart_send_break	send break frame
usart_halfduplex_enable	enable half duplex mode
usart_halfduplex_disable	disable half duplex mode
usart_synchronous_clock_enable	enable CK pin in synchronous mode
usart_synchronous_clock_disable	disable CK pin in synchronous mode

Function name	Function description
usart_synchronous_clock_config	configure USART synchronous mode parameters
usart_guard_time_config	configure guard time value in smartcard mode
usart_smartcard_mode_enable	enable smartcard mode
usart_smartcard_mode_disable	disable smartcard mode
usart_smartcard_mode_nack_enable	enable NACK in smartcard mode
usart_smartcard_mode_nack_disable	disable NACK in smartcard mode
usart_smartcard_autoretry_config	configure smartcard auto-retry number
usart_block_length_config	configure block length
usart_irda_mode_enable	enable IrDA mode
usart_irda_mode_disable	disable IrDA mode
usart_prescaler_config	configure the peripheral clock prescaler in USART IrDA low-power 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_dma_receive_config	configure USART DMA reception
usart_dma_transmit_config	configure USART DMA transmission
usart_flag_get	get flag in STAT0/STAT1 register
usart_flag_clear	clear flag in STAT0/STAT1 register
usart_interrupt_enable	enable USART interrupt
usart_interrupt_disable	disable USART interrupt
usart_interrupt_flag_get	get USART interrupt flag status
usart_interrupt_flag_clear	clear USART interrupt flag

## usart\_deinit

The description of usart\_deinit is shown as below:

**Table 3-614. Function usart\_deinit**

Function name	usart_deinit
---------------	--------------

<b>Function prototype</b>	void usart_deinit(uint32_t usart_periph);
<b>Function descriptions</b>	reset USART/UART
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<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 3-615. 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_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	

<b>baudval</b>	baud rate value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 baud rate value */
```

```
usart_baudrate_set(USART0, 115200);
```

### usart\_parity\_config

The description of usart\_parity\_config is shown as below:

**Table 3-616. Function usart\_parity\_config**

<b>Function name</b>	usart_parity_config
<b>Function prototype</b>	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
<b>Function descriptions</b>	configure USART parity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>paritycfg</b>	configure USART parity
<i>USART_PM_NONE</i>	no parity
<i>USART_PM_ODD</i>	odd parity
<i>USART_PM_EVEN</i>	even parity
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* configure USART parity */
```

```
usart_parity_config(USART0, USART_PM_EVEN);
```

### usart\_word\_length\_set

The description of usart\_word\_length\_set is shown as below:

**Table 3-617. Function usart\_word\_length\_set**

<b>Function name</b>	usart_word_length_set
<b>Function prototype</b>	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
<b>Function descriptions</b>	configure USART word length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>wlen</b>	USART word length
<i>USART_WL_8BIT</i>	8 bits
<i>USART_WL_9BIT</i>	9 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 word length */
```

```
usart_word_length_set(USART0, USART_WL_9BIT);
```



## usart\_stop\_bit\_set

The description of usart\_stop\_bit\_set is shown as below:

**Table 3-618. Function usart\_stop\_bit\_set**

<b>Function name</b>	usart_stop_bit_set
<b>Function prototype</b>	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
<b>Function descriptions</b>	configure USART stop bit length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>stblen</b>	USART stop bit
<i>USART_STB_1BIT</i>	1 bit
<i>USART_STB_0_5BIT</i>	0.5 bit, not available for UARTx(x=3,4)
<i>USART_STB_2BIT</i>	2 bits
<i>USART_STB_1_5BIT</i>	1.5 bits, not available for UARTx(x=3,4)
<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 3-619. Function `usart_enable`

Function name	<code>usart_enable</code>
Function prototype	<code>void usart_enable(uint32_t usart_periph);</code>
Function descriptions	enable USART
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	x=0,1,2
<code>UARTx</code>	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 */
usart_enable(USART0);
```

### **`usart_disable`**

The description of `usart_disable` is shown as below:

Table 3-620. Function `usart_disable`

Function name	<code>usart_disable</code>
Function prototype	<code>void usart_disable(uint32_t usart_periph);</code>
Function descriptions	disable USART
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	x=0,1,2

<i>UARTx</i>	x=3,4
<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 3-621. Function usart\_transmit\_config**

<b>Function name</b>	usart_transmit_config
<b>Function prototype</b>	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
<b>Function descriptions</b>	configure USART transmitter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>txconfig</b>	enable or disable USART transmitter
<i>USART_TRANSMIT_ENABLE</i>	enable USART transmission
<i>USART_TRANSMIT_DISABLE</i>	enable 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 3-622. Function usart\_receive\_config**

<b>Function name</b>	usart_receive_config
<b>Function prototype</b>	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
<b>Function descriptions</b>	configure USART receiver
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>rxconfig</b>	enable or disable USART receiver
<i>USART_RECEIVE_ENABLE</i>	enable USART reception
<i>USART_RECEIVE_DISABLE</i>	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 3-623. 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>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>msbf</b>	LSB first or MSB first
<i>USART_MSBF_LSB</i>	LSB first
<i>USART_MSBF_MSB</i>	MSB first
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LSB of data first */
```

```
usart_data_first_config(USART0, USART_MSBF_LSB);
```

## usart\_invert\_config

The description of usart\_invert\_config is shown as below:

**Table 3-624. 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>	configure USART inversion
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>invertpara</b>	refer to enum usart_invert_enum
<i>USART_DINV_ENAB LE</i>	data bit level inversion
<i>USART_DINV_DISAB LE</i>	data bit level not inversion
<i>USART_TXPIN_ENAB LE</i>	TX pin level inversion
<i>USART_TXPIN_DISAB LE</i>	TX pin level not inversion
<i>USART_RXPIN_ENAB LE</i>	RX pin level inversion
<i>USART_RXPIN_DISAB LE</i>	RX pin level not inversion
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART inversion */
```

```
usart_invert_config(USART0, USART_DINV_ENABLE);
```

### **usart\_receiver\_timeout\_enable**

The description of usart\_receiver\_timeout\_enable is shown as below:

Table 3-625. Function `usart_receiver_timeout_enable`

Function name	<code>usart_receiver_timeout_enable</code>
Function prototype	<code>void usart_receiver_timeout_enable(uint32_t usart_periph);</code>
Function descriptions	enable receiver timeout
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx peripheral
<code>USARTx</code>	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable receiver timeout of USART */
usart_receiver_timeout_enable(USART0);
```

### `usart_receiver_timeout_disable`

The description of `usart_receiver_timeout_disable` is shown as below:

Table 3-626. Function `usart_receiver_timeout_disable`

Function name	<code>usart_receiver_timeout_disable</code>
Function prototype	<code>void usart_receiver_timeout_disable(uint32_t usart_periph);</code>
Function descriptions	disable receiver timeout
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx peripheral
<code>USARTx</code>	x=0,1,2
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 3-627. Function usart\_receiver\_timeout\_threshold\_config**

<b>Function name</b>	usart_receiver_timeout_threshold_config
<b>Function prototype</b>	void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);
<b>Function descriptions</b>	configure receiver timeout threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>rtimeout</b>	timeout value
<i>0-0xFFFFFFFF</i>	timeout value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the receiver timeout threshold of USART0 */
```

```
usart_receiver_timeout_threshold_config(USART0, 115200*3);
```



## usart\_data\_transmit

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

**Table 3-628. Function usart\_data\_transmit**

<b>Function name</b>	usart_data_transmit
<b>Function prototype</b>	void usart_data_transmit(uint32_t usart_periph, uint32_t data);
<b>Function descriptions</b>	USART transmit data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>data</b>	data of transmission
<i>0-0xFF</i>	data of transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 transmit data */
usart_data_transmit(USART0, 0xAA);
```

## usart\_data\_receive

The description of usart\_data\_receive is shown as below:

**Table 3-629. Function usart\_data\_receive**

<b>Function name</b>	usart_data_receive
<b>Function prototype</b>	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
<b>Function descriptions</b>	USART receive data function

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	data of received(0-0x1FF)

Example:

```
/* USART0 receive data */
```

```
uint16_t temp;
```

```
temp = usart_data_receive(USART0);
```

### usart\_address\_config

The description of usart\_address\_config is shown as below:

**Table 3-630. Function usart\_address\_config**

<b>Function name</b>	usart_address_config
<b>Function prototype</b>	void usart_address_config(uint32_t usart_periph, uint8_t addr);
<b>Function descriptions</b>	configure the address of the USART in wake up by address match mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>addr</b>	address of USART/UART

0-0xFF	address of USART/UART
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure address of the USART0 */
```

```
usart_address_config(USART0, 0x00);
```

### usart\_mute\_mode\_enable

The description of usart\_mute\_mode\_enable is shown as below:

**Table 3-631. Function usart\_mute\_mode\_enable**

<b>Function name</b>	usart_mute_mode_enable
<b>Function prototype</b>	void usart_mute_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 receiver in mute mode */
```

```
usart_mute_mode_enable(USART0);
```

## usart\_mute\_mode\_disable

The description of usart\_mute\_mode\_disable is shown as below:

**Table 3-632. Function usart\_mute\_mode\_disable**

<b>Function name</b>	usart_mute_mode_disable
<b>Function prototype</b>	void usart_mute_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<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 3-633. 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>	-

Input parameter{in}	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
Input parameter{in}	
<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 mask
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 wakeup method in mute mode */
```

```
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

### usart\_lin\_mode\_enable

The description of usart\_lin\_mode\_enable is shown as below:

**Table 3-634. 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>	-
Input parameter{in}	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
Output parameter{out}	

-	-
Return value	
-	-

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 3-635. Function usart\_lin\_mode\_disable**

<b>Function name</b>	usart_lin_mode_disable
<b>Function prototype</b>	void usart_lin_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable LIN mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<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 3-636. Function `usart_lin_break_dection_length_config`

<b>Function name</b>	<code>usart_lin_break_dection_length_config</code>
<b>Function prototype</b>	<code>void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lblen);</code>
<b>Function descriptions</b>	configure LIN break frame length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>lblen</b>	two methods be used to enter or exit the mute mode
<i>USART_LBLEN_10B</i>	break frame length is 10 bits
<i>USART_LBLEN_11B</i>	break frame length is 11 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LIN break frame length */
```

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

### **usart\_send\_break**

The description of `usart_send_break` is shown as below:

Table 3-637. Function `usart_send_break`

<b>Function name</b>	<code>usart_send_break</code>
<b>Function prototype</b>	<code>void usart_send_break(uint32_t usart_periph);</code>
<b>Function descriptions</b>	send break frame
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 send break frame */
```

```
usart_send_break(USART0);
```

### usart\_halfduplex\_enable

The description of usart\_halfduplex\_enable is shown as below:

**Table 3-638. Function usart\_halfduplex\_enable**

<b>Function name</b>	usart_halfduplex_enable
<b>Function prototype</b>	void usart_halfduplex_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable half duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<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 3-639. Function usart\_halfduplex\_disable**

<b>Function name</b>	usart_halfduplex_disable
<b>Function prototype</b>	void usart_halfduplex_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable half duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 half duplex mode*/
```

```
usart_halfduplex_disable(USART0);
```

### usart\_synchronous\_clock\_enable

The description of usart\_synchronous\_clock\_enable is shown as below:

**Table 3-640. Function usart\_synchronous\_clock\_enable**

<b>Function name</b>	usart_synchronous_clock_enable
<b>Function prototype</b>	void usart_synchronous_clock_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable CK pin in synchronous mode

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 CK pin in synchronous mode */
```

```
usart_synchronous_clock_enable(USART0);
```

### **usart\_synchronous\_clock\_disable**

The description of usart\_synchronous\_clock\_disable is shown as below:

**Table 3-641. Function usart\_synchronous\_clock\_disable**

<b>Function name</b>	usart_synchronous_clock_disable
<b>Function prototype</b>	void usart_synchronous_clock_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable CK pin in synchronous mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 CK pin in synchronous mode */
```

```
usart_synchronous_clock_disable(USART0);
```

### usart\_synchronous\_clock\_config

The description of usart\_synchronous\_clock\_config is shown as below:

**Table 3-642. Function usart\_synchronous\_clock\_config**

<b>Function name</b>	usart_synchronous_clock_config
<b>Function prototype</b>	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
<b>Function descriptions</b>	configure USART synchronous mode parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>clen</b>	CK length
<i>USART_CLEN_NONE</i>	there are 7 CK pulses for an 8 bit frame and 8 CK pulses for a 9 bit frame
<i>USART_CLEN_EN</i>	there are 8 CK pulses for an 8 bit frame and 9 CK pulses for a 9 bit frame
<b>Input parameter{in}</b>	
<b>cph</b>	clock phase
<i>USART_CPH_1CK</i>	first clock transition is the first data capture edge
<i>USART_CPH_2CK</i>	second clock transition is the first data capture edge
<b>Input parameter{in}</b>	
<b>cpl</b>	clock polarity
<i>USART_CPL_LOW</i>	steady low value on CK pin
<i>USART_CPL_HIGH</i>	steady high value on CK pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* configure USART0 synchronous mode parameters */
```

```
usart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,
USART_CPL_HIGH);
```

### usart\_guard\_time\_config

The description of usart\_guard\_time\_config is shown as below:

**Table 3-643. Function usart\_guard\_time\_config**

<b>Function name</b>	usart_guard_time_config
<b>Function prototype</b>	void usart_guard_time_config(uint32_t usart_periph,uint32_t gaut);
<b>Function descriptions</b>	configure guard time value in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>gaut</b>	guard time value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 guard time value in smartcard mode */
```

```
usart_guard_time_config(USART0, 0x0000 0055);
```

### usart\_smartcard\_mode\_enable

The description of usart\_smartcard\_mode\_enable is shown as below:

Table 3-644. Function `usart_smartcard_mode_enable`

Function name	<code>usart_smartcard_mode_enable</code>
Function prototype	<code>void usart_smartcard_mode_enable(uint32_t usart_periph);</code>
Function descriptions	enable smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx peripheral
<code>USARTx</code>	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

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 3-645. Function `usart_smartcard_mode_disable`

Function name	<code>usart_smartcard_mode_disable</code>
Function prototype	<code>void usart_smartcard_mode_disable(uint32_t usart_periph);</code>
Function descriptions	disable smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx peripheral
<code>USARTx</code>	x=0,1,2
Output parameter{out}	

-	-
Return value	
-	-

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 3-646. Function usart\_smartcard\_mode\_nack\_enable**

<b>Function name</b>	usart_smartcard_mode_nack_enable
<b>Function prototype</b>	void usart_smartcard_mode_nack_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 NACK in smartcard mode */
```

```
usart_smartcard_mode_nack_enable(USART0);
```

### usart\_smartcard\_mode\_nack\_disable

The description of usart\_smartcard\_mode\_nack\_disable is shown as below:

**Table 3-647. Function usart\_smartcard\_mode\_nack\_disable**

<b>Function name</b>	usart_smartcard_mode_nack_disable
----------------------	-----------------------------------

<b>Function prototype</b>	void usart_smartcard_mode_nack_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 NACK in smartcard mode */
```

```
usart_smartcard_mode_nack_disable(USART0);
```

### usart\_smartcard\_autoretry\_config

The description of usart\_smartcard\_autoretry\_config is shown as below:

**Table 3-648. Function usart\_smartcard\_autoretry\_config**

<b>Function name</b>	usart_smartcard_autoretry_config
<b>Function prototype</b>	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);
<b>Function descriptions</b>	configure smartcard auto-retry number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>scrtnum</b>	smartcard auto-retry number

0-0xFFFFFFFF	smartcard auto-retry number
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure smartcard auto-retry number */
```

```
usart_smartcard_autoretry_config (USART0, 0xFFFFFFFF);
```

### usart\_block\_length\_config

The description of usart\_block\_length\_config is shown as below:

**Table 3-649. 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>	-
Input parameter{in}	
<b>usart_periph</b>	USARTx peripheral
USARTx	x=0,1,2
Input parameter{in}	
<b>bl</b>	block length
0-0xFFFFFFFF	block length
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure block length in Smartcard T=1 reception */
```



```
usart_block_length_config(USART0, 0xFFFFFFFF);
```

### usart\_irda\_mode\_enable

The description of usart\_irda\_mode\_enable is shown as below:

**Table 3-650. Function usart\_irda\_mode\_enable**

<b>Function name</b>	usart_irda_mode_enable
<b>Function prototype</b>	void usart_irda_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable IrDA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<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 3-651. 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>	-

Input parameter{in}	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 IrDA mode */
```

```
usart_irda_mode_disable(USART0);
```

### usart\_prescaler\_config

The description of usart\_prescaler\_config is shown as below:

**Table 3-652. Function usart\_prescaler\_config**

<b>Function name</b>	usart_prescaler_config
<b>Function prototype</b>	void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);
<b>Function descriptions</b>	configure the peripheral clock prescaler in USART IrDA low-power mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
Input parameter{in}	
<b>psc</b>	0x00-0xFF
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure the USART0 peripheral clock prescaler */
```

```
usart_prescaler_config(USART0, 0x00);
```

### usart\_irda\_lowpower\_config

The description of usart\_irda\_lowpower\_config is shown as below:

**Table 3-653. Function usart\_irda\_lowpower\_config**

<b>Function name</b>	usart_irda_lowpower_config
<b>Function prototype</b>	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
<b>Function descriptions</b>	configure IrDA low-power
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<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 3-654. Function usart\_hardware\_flow\_rts\_config**

<b>Function name</b>	usart_hardware_flow_rts_config
<b>Function prototype</b>	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
<b>Function descriptions</b>	configure hardware flow control RTS
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>rtsconfig</b>	enable or disable RTS
<i>USART_RTS_ENABLE</i>	enable RTS
<i>USART_RTS_DISABLE</i>	disable RTS
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 hardware flow control RTS */
```

```
usart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

## usart\_hardware\_flow\_cts\_config

The description of usart\_hardware\_flow\_cts\_config is shown as below:

**Table 3-655. 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 RTS
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>ctsconfig</b>	enable or disable CTS
<i>USART_CTS_ENABLE</i>	enable CTS
<i>USART_CTS_DISABLE</i>	disable CTS
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 hardware flow control CTS */
```

```
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

### usart\_dma\_receive\_config

The description of usart\_dma\_receive\_config is shown as below:

**Table 3-656. 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, uint32_t dmacmd);
<b>Function descriptions</b>	configure USART DMA reception
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral

<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3
<b>Input parameter{in}</b>	
<b>dmacmd</b>	enable or disable DMA for reception
<i>USART_RECEIVE_DMA_ENABLE</i>	DMA enable for reception
<i>USART_RECEIVE_DMA_DISABLE</i>	DMA disable for reception
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 DMA enable for reception */
```

```
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

### usart\_dma\_transmit\_config

The description of usart\_dma\_transmit\_config is shown as below:

**Table 3-657. 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, uint32_t dmacmd);
<b>Function descriptions</b>	configure USART DMA transmission
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3
<b>Input parameter{in}</b>	
<b>dmacmd</b>	enable or disable DMA for transmission

<i>USART_TRANSMIT_DMA_ENABLE</i>	DMA enable for transmission
<i>USART_TRANSMIT_DMA_DISABLE</i>	DMA disable for transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 DMA enable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

### usart\_flag\_get

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

**Table 3-658. 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 STAT0/STAT1 register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to usart_flag_enum
<i>USART_FLAG_CTS</i>	CTS change flag
<i>USART_FLAG_LBD</i>	LIN break detected flag
<i>USART_FLAG_TBE</i>	transmit data buffer empty flag
<i>USART_FLAG_TC</i>	transmission complete flag

<i>USART_FLAG_RBNE</i>	read data buffer not empty flag
<i>USART_FLAG_IDLE</i>	IDLE frame detected flag
<i>USART_FLAG_ORER</i> <i>R</i>	overrun error flag
<i>USART_FLAG_NERR</i>	noise error flag
<i>USART_FLAG_FERR</i>	frame error flag
<i>USART_FLAG_PERR</i>	parity error flag
<i>USART_FLAG_BSY</i>	busy flag
<i>USART_FLAG_EB</i>	end of block flag
<i>USART_FLAG_RT</i>	receiver timeout flag
<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 3-659. 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 flag in STAT0/STAT1 register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral



<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to usart_flag_enum
<i>USART_FLAG_CTS</i>	CTS change flag
<i>USART_FLAG_LBD</i>	LIN break detected flag
<i>USART_FLAG_TC</i>	transmission complete
<i>USART_FLAG_RBNE</i>	read data buffer not empty
<i>USART_FLAG_EB</i>	end of block flag
<i>USART_FLAG_RT</i>	receiver timeout flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear USART0 flag */
```

```
usart_flag_clear(USART0,USART_FLAG_TC);
```

### usart\_interrupt\_enable

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

**Table 3-660. Function usart\_interrupt\_enable**

<b>Function name</b>	usart_interrupt_enable
<b>Function prototype</b>	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
<b>Function descriptions</b>	enable USART interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral

<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>interrupt</b>	USART interrupt
<i>USART_INT_PERR</i>	parity error interrupt
<i>USART_INT_TBE</i>	transmitter buffer empty interrupt
<i>USART_INT_TC</i>	transmission complete interrupt
<i>USART_INT_RBNE</i>	read data buffer not empty interrupt and overrun error interrupt
<i>USART_INT_IDLE</i>	IDLE line detected interrupt
<i>USART_INT_LBD</i>	LIN break detected interrupt
<i>USART_INT_ERR</i>	error interrupt
<i>USART_INT_CTS</i>	CTS interrupt
<i>USART_INT_RT</i>	receive timeout event interrupt
<i>USART_INT_EB</i>	end of block event interrupt
<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 3-661. Function usart\_interrupt\_disable**

<b>Function name</b>	usart_interrupt_disable
<b>Function prototype</b>	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
<b>Function descriptions</b>	disable USART interrupt

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag
<i>USART_INT_PERR</i>	parity error interrupt
<i>USART_INT_TBE</i>	transmitter buffer empty interrupt
<i>USART_INT_TC</i>	transmission complete interrupt
<i>USART_INT_RBNE</i>	read data buffer not empty interrupt and overrun error interrupt
<i>USART_INT_IDLE</i>	IDLE line detected interrupt
<i>USART_INT_LBD</i>	LIN break detected interrupt
<i>USART_INT_ERR</i>	error interrupt
<i>USART_INT_CTS</i>	CTS interrupt
<i>USART_INT_RT</i>	receive timeout event interrupt
<i>USART_INT_EB</i>	end of block event interrupt
<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 3-662. Function `usart_interrupt_flag_get`

<b>Function name</b>	<code>usart_interrupt_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);</code>
<b>Function descriptions</b>	get USART interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag, refer to <code>usart_interrupt_flag_enum</code>
<i>USART_INT_FLAG_PERRR</i>	parity error interrupt and flag
<i>USART_INT_FLAG_TB E</i>	transmitter buffer empty interrupt and flag
<i>USART_INT_FLAG_TC</i>	transmission complete interrupt and flag
<i>USART_INT_FLAG_RB NE</i>	read data buffer not empty interrupt and flag
<i>USART_INT_FLAG_RB NE_ORERR</i>	read data buffer not empty interrupt and overrun error flag
<i>USART_INT_FLAG_ID LE</i>	IDLE line detected interrupt and flag
<i>USART_INT_FLAG_LB D</i>	LIN break detected interrupt and flag
<i>USART_INT_FLAG_CT S</i>	CTS interrupt and flag
<i>USART_INT_FLAG_ER R_ORERR</i>	error interrupt and overrun error
<i>USART_INT_FLAG_ER R_NERR</i>	error interrupt and noise error flag

<i>USART_INT_FLAG_ER R_FERR</i>	error interrupt and frame error flag
<i>USART_INT_FLAG_EB</i>	end of block event interrupt flag
<i>USART_INT_FLAG_RT</i>	receive timeout event interrupt flag
<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 3-663. 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>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag, refer to usart_interrupt_flag_enum
<i>USART_INT_FLAG_CTS</i>	CTS change flag

<i>USART_INT_FLAG_LB D</i>	LIN break detected flag
<i>USART_INT_FLAG_TC</i>	transmission complete
<i>USART_INT_FLAG_RB NE</i>	read data buffer not empty
<i>USART_INT_FLAG_EB</i>	end of block event interrupt flag
<i>USART_INT_FLAG_RT</i>	receive timeout event interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the USART0 interrupt enable bit status */
```

```
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

## 3.24. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.24.1](#), the WWDGT firmware functions are introduced in chapter [3.24.2](#).

### 3.24.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

**Table 3-664. WWDGT Registers**

Registers	Descriptions
WWDGT_CTL	Control register
WWDGT_CFG	Configuration register
WWDGT_STAT	Status register

### 3.24.2. Descriptions of Peripheral functions

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

Table 3-665. WWDGT firmware function

Function name	Function description
wwdgt_deinit	reset the window watchdog timer configuration
wwdgt_enable	start the window watchdog timer counter
wwdgt_counter_update	configure the window watchdog timer counter value
wwdgt_config	configure counter value, window value, and prescaler divider value
wwdgt_interrupt_enable	enable early wakeup interrupt of WWDGT
wwdgt_flag_get	check early wakeup interrupt state of WWDGT
wwdgt_flag_clear	clear early wakeup interrupt state of WWDGT

### wwdgt\_deinit

The description of wwdgt\_deinit is shown as below:

Table 3-666. Function wwdgt\_deinit

Function name	wwdgt_deinit
Function prototype	void wwdgt_deinit(void);
Function descriptions	reset the window watchdog timer configuration
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the window watchdog timer configuration */
```

```
wwdgt_deinit ( );
```

## wwdgt\_enable

The description of wwdgt\_enable is shown as below:

**Table 3-667. Function wwdgt\_enable**

<b>Function name</b>	wwdgt_enable
<b>Function prototype</b>	void wwdgt_enable (void);
<b>Function descriptions</b>	start the window watchdog timer 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 window watchdog timer counter */
```

```
wwdgt_enable ( );
```

## wwdgt\_counter\_update

The description of wwdgt\_counter\_update is shown as below:

**Table 3-668. Function wwdgt\_counter\_update**

<b>Function name</b>	wwdgt_counter_update
<b>Function prototype</b>	void wwdgt_counter_update(uint16_t counter_value);
<b>Function descriptions</b>	configure the window watchdog timer counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
counter_value	0x00 - 0x7F
<b>Output parameter{out}</b>	



-	-
Return value	
-	-

Example:

```
/* update WWDGT counter to 0x7F */
```

```
wwdgt_counter_update(127);
```

### wwdgt\_config

The description of wwdgt\_config is shown as below:

**Table 3-669. Function wwdgt\_config**

<b>Function name</b>	wwdgt_config
<b>Function prototype</b>	void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);
<b>Function descriptions</b>	configure counter value, window value, and prescaler divider value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>counter</b>	0x00 - 0x7F
<b>Input parameter{in}</b>	
<b>window</b>	0x00 - 0x7F
<b>Input parameter{in}</b>	
<b>prescaler</b>	wwdgt prescaler value
WWDGT_CFG_PSC_D IV1	the time base of window watchdog counter = (PCLK1/4096)/1
WWDGT_CFG_PSC_D IV2	the time base of window watchdog counter = (PCLK1/4096)/2
WWDGT_CFG_PSC_D IV4	the time base of window watchdog counter = (PCLK1/4096)/4
WWDGT_CFG_PSC_D IV8	the time base of window watchdog counter = (PCLK1/4096)/8
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

### wwdgt\_interrupt\_enable

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

**Table 3-670. Function wwdgt\_interrupt\_enable**

Function name	wwdgt_interrupt_enable
Function prototype	void wwdgt_interrupt_enable(void);
Function descriptions	enable early wakeup interrupt of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable early wakeup interrupt of WWDGT */
```

```
wwdgt_interrupt_enable ( );
```

### wwdgt\_flag\_get

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

**Table 3-671. Function wwdgt\_flag\_get**

Function name	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 ( );
```

```
if(status == RESET)
```

```
{
```

```
...
```

```
}else
```

```
{
```

```
...
```

```
}
```

### wwdgt\_flag\_clear

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

**Table 3-672. 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	Dec.26, 2018
1.1	Format update	July.7, 2022
1.2	Consistency update	Mar.30, 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.