

**GigaDevice Semiconductor Inc.**

**GD32F50x**

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

**Firmware Library  
User Guide**

Revision 1.1

(Feb. 2026)

# 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>26</b>
<b>1.1. Rules of User Manual and Firmware Library .....</b>	<b>26</b>
1.1.1. Peripherals.....	26
1.1.2. Naming rules.....	27
<b>2. Firmware Library Overview .....</b>	<b>29</b>
<b>2.1. File Structure of Firmware Library .....</b>	<b>29</b>
2.1.1. Examples Folder .....	30
2.1.2. Firmware Folder .....	30
2.1.3. Template Folder .....	30
2.1.4. Utilities Folder .....	32
<b>2.2. File descriptions of Firmware Library .....</b>	<b>33</b>
<b>3. Firmware Library of Standard Peripherals .....</b>	<b>34</b>
<b>3.1. Overview of Firmware Library of Standard Peripherals.....</b>	<b>34</b>
<b>3.2. ADC .....</b>	<b>34</b>
3.2.1. Descriptions of Peripheral registers.....	34
3.2.2. Descriptions of Peripheral functions .....	35
<b>3.3. BKP.....</b>	<b>60</b>
3.3.1. Descriptions of Peripheral registers.....	60
3.3.2. Descriptions of Peripheral functions .....	61
<b>3.4. CAN .....</b>	<b>75</b>
3.4.1. Descriptions of Peripheral registers.....	75
3.4.2. Descriptions of Peripheral functions .....	76
<b>3.5. CAU .....</b>	<b>99</b>
3.5.1. Descriptions of Peripheral registers.....	99
3.5.2. Descriptions of Peripheral functions .....	99
<b>3.6. CMP .....</b>	<b>111</b>
3.6.1. Descriptions of Peripheral registers.....	111
3.6.2. Descriptions of Peripheral functions .....	112
<b>3.7. CRC .....</b>	<b>124</b>
3.7.1. Descriptions of Peripheral registers.....	124
3.7.2. Descriptions of Peripheral functions .....	124

<b>3.8. CTC</b>	<b>132</b>
3.8.1. Descriptions of Peripheral registers	132
3.8.2. Descriptions of Peripheral functions	132
<b>3.9. DAC</b>	<b>145</b>
3.9.1. Peripheral register description	145
3.9.2. Descriptions of Peripheral functions	145
<b>3.10. DBG</b>	<b>161</b>
3.10.1. Descriptions of Peripheral registers	161
3.10.2. Descriptions of Peripheral functions	162
<b>3.11. DMA / DMAMUX</b>	<b>168</b>
3.11.1. Descriptions of Peripheral registers	168
3.11.2. Descriptions of Peripheral functions	169
<b>3.12. EXMC</b>	<b>214</b>
3.12.1. Descriptions of Peripheral registers	215
3.12.2. Descriptions of Peripheral functions	215
<b>3.13. EXTI</b>	<b>229</b>
3.13.1. Descriptions of Peripheral registers	229
3.13.2. Descriptions of Peripheral functions	229
<b>3.14. FMC</b>	<b>236</b>
3.14.1. Descriptions of Peripheral registers	236
3.14.2. Descriptions of Peripheral functions	237
<b>3.15. FWDGT</b>	<b>265</b>
3.15.1. Descriptions of Peripheral registers	266
3.15.2. Descriptions of Peripheral functions	266
<b>3.16. GPIO</b>	<b>271</b>
3.16.1. Descriptions of Peripheral registers	271
3.16.2. Descriptions of Peripheral functions	272
<b>3.17. HAU</b>	<b>282</b>
3.17.1. Descriptions of Peripheral registers	282
3.17.2. Descriptions of Peripheral functions	282
<b>3.18. I2C</b>	<b>293</b>
3.18.1. Descriptions of Peripheral registers	293
3.18.2. Descriptions of Peripheral functions	293
<b>3.19. MISC</b>	<b>331</b>
3.19.1. Descriptions of Peripheral registers	331
3.19.2. Descriptions of Peripheral functions	332
<b>3.20. PMU</b>	<b>339</b>
3.20.1. Descriptions of Peripheral registers	339
3.20.2. Descriptions of Peripheral functions	339

<b>3.21.</b>	<b>RCU .....</b>	<b>360</b>
3.21.1.	Descriptions of Peripheral registers .....	360
3.21.2.	Descriptions of Peripheral functions .....	361
<b>3.22.</b>	<b>RTC .....</b>	<b>406</b>
3.22.1.	Descriptions of Peripheral registers .....	406
3.22.2.	Descriptions of Peripheral functions .....	407
<b>3.23.</b>	<b>SYSCFG .....</b>	<b>415</b>
3.23.1.	Descriptions of Peripheral registers .....	415
3.23.2.	Descriptions of Peripheral functions .....	416
<b>3.24.</b>	<b>SPI .....</b>	<b>430</b>
3.24.1.	Descriptions of Peripheral registers .....	430
3.24.2.	Descriptions of Peripheral functions .....	430
<b>3.25.</b>	<b>TIMER .....</b>	<b>455</b>
3.25.1.	Descriptions of Peripheral registers .....	455
3.25.2.	Descriptions of Peripheral functions .....	456
<b>3.26.</b>	<b>TRIGSEL .....</b>	<b>543</b>
3.26.1.	Descriptions of Peripheral registers .....	543
3.26.2.	Descriptions of Peripheral functions .....	544
<b>3.27.</b>	<b>TRNG .....</b>	<b>552</b>
3.27.1.	Descriptions of Peripheral registers .....	552
3.27.2.	Descriptions of Peripheral functions .....	552
<b>3.28.</b>	<b>USART .....</b>	<b>558</b>
3.28.1.	Descriptions of Peripheral registers .....	558
3.28.2.	Descriptions of Peripheral functions .....	558
<b>3.29.</b>	<b>WWDGT .....</b>	<b>594</b>
3.29.1.	Descriptions of Peripheral registers .....	594
3.29.2.	Descriptions of Peripheral functions .....	594
<b>4.</b>	<b>Revision history .....</b>	<b>600</b>



## List of Figures

Figure 2-1. File structure of firmware library of GD32F50x .....	29
Figure 2-2. Select peripheral example files .....	31
Figure 2-3. Copy the peripheral example files .....	31
Figure 2-4. Open the project file .....	31
Figure 2-5. Configure project files .....	32
Figure 2-6. Compile-debug-download .....	32

# List of Tables

Table 1-1. Peripherals .....	26
Table 2-1. Function descriptions of Firmware Library .....	33
Table 3-1. Peripheral function format of Firmware Library .....	34
Table 3-2. ADC Registers .....	34
Table 3-3. ADC firmware function.....	35
Table 3-4. Function adc_deinit.....	36
Table 3-5. Function adc_enable .....	36
Table 3-6. Function adc_disable .....	37
Table 3-7. Function adc_dma_mode_enable .....	37
Table 3-8. Function adc_dma_mode_disable.....	38
Table 3-9. Function adc_discontinuous_mode_config .....	39
Table 3-10. Function adc_special_function_config .....	39
Table 3-11. Function adc_internal_channel_config .....	40
Table 3-12. Function adc_sync_mode_config.....	41
Table 3-13. Function adc_sync_routine_data_read .....	42
Table 3-14. Function adc_data_alignment_config.....	42
Table 3-15. Function adc_channel_length_config.....	43
Table 3-16. Function adc_routine_channel_config .....	44
Table 3-17. Function adc_inserted_channel_config .....	45
Table 3-18. Function adc_inserted_channel_offset_config .....	46
Table 3-19. Function adc_external_trigger_config.....	47
Table 3-20. Function adc_software_trigger_enable .....	47
Table 3-21. Function adc_routine_data_read .....	48
Table 3-22. Function adc_inserted_data_read .....	49
Table 3-23. Function adc_latch_data_read .....	49
Table 3-24. Function adc_latch_data_source_config .....	50
Table 3-25. Function adc_watchdog_single_channel_enable.....	50
Table 3-26. Function adc_watchdog0_sequence_channel_enable .....	51
Table 3-27. Function adc_watchdog_disable .....	52
Table 3-28. Function adc_watchdog0_threshold_config .....	52
Table 3-29. Function adc_resolution_config .....	53
Table 3-30. Function adc_oversample_mode_config .....	54
Table 3-31. Function adc_oversample_mode_enable .....	55
Table 3-32. Function adc_oversample_mode_disable .....	56
Table 3-33. Function adc_flag_get .....	56
Table 3-34. Function adc_flag_clear .....	57
Table 3-35. Function adc_interrupt_enable .....	57
Table 3-36. Function adc_interrupt_disable .....	58
Table 3-37. Function adc_interrupt_flag_get.....	59
Table 3-38. Function adc_interrupt_flag_clear .....	59
Table 3-39. BKP Registers .....	60

Table 3-40. BKP firmware function.....	61
Table 3-41. Function bkp_deinit.....	61
Table 3-42. Function bkp_write_data.....	62
Table 3-43. Function bkp_data_read.....	63
Table 3-44. Function bkp_rtc_calibration_output_enable.....	64
Table 3-45. Function bkp_rtc_calibration_output_disable.....	64
Table 3-46. Function bkp_rtc_signal_output_enable.....	65
Table 3-47. Function bkp_rtc_signal_output_disable.....	65
Table 3-48. Function bkp_rtc_output_select.....	66
Table 3-49. Function bkp_rtc_clock_output_select.....	67
Table 3-50. Function bkp_rtc_clock_calibration_direction.....	67
Table 3-51. Function bkp_rtc_calibration_value_set.....	68
Table 3-52. Function bkp_tamper_detection_enable.....	69
Table 3-53. Function bkp_tamper_detection_disable.....	69
Table 3-54. Function bkp_tamper_active_level_set.....	70
Table 3-55. Function bkp_tamper_interrupt_enable.....	71
Table 3-56. Function bkp_tamper_interrupt_disable.....	71
Table 3-57. Function bkp_flag_get.....	72
Table 3-58. Function bkp_flag_clear.....	73
Table 3-59. Function bkp_interrupt_flag_get.....	73
Table 3-60. Function bkp_interrupt_flag_clear.....	74
Table 3-61. CAN Registers.....	75
Table 3-62. CAN firmware function.....	76
Table 3-63. Structure can_parameter_struct.....	76
Table 3-64. Structure can_trasnmit_message_struct.....	77
Table 3-65. Structure can_receive_message_struct.....	77
Table 3-66. Structure can_filter_parameter_struct.....	78
Table 3-67. Structure can_fd_tdc_struct.....	78
Table 3-68. Structure can_fdframe_struct.....	78
Table 3-69. Function can_deinit.....	79
Table 3-70. Function can_struct_para_init.....	79
Table 3-71. Function can_init.....	80
Table 3-72. Function can_fd_init.....	80
Table 3-73. Function can_filter_init.....	81
Table 3-74. Function can_filter_mask_mode_init.....	82
Table 3-75. Function can_monitor_mode_set.....	82
Table 3-76. Function can_fd_function_enable.....	83
Table 3-77. Function can_fd_function_disable.....	84
Table 3-78. Function can1_filter_start_bank.....	84
Table 3-79. Function can_debug_freeze_enable.....	85
Table 3-80. Function can_debug_freeze_disable.....	85
Table 3-81. Function can_time_trigger_mode_enable.....	86
Table 3-82. Function can_time_trigger_mode_disable.....	86
Table 3-83. Function can_message_transmit.....	87

Table 3-84. Function can_transmit_states .....	87
Table 3-85. Function can_transmission_stop .....	88
Table 3-86. Function can_message_receive .....	88
Table 3-87. Function can_fifo_release .....	89
Table 3-88. Function can_receive_message_length_get .....	90
Table 3-89. Function can_working_mode_set.....	90
Table 3-90. Function can_wakeup .....	91
Table 3-91. Function can_error_get .....	92
Table 3-92. Function can_receive_error_number_get .....	92
Table 3-93. Function can_transmit_error_number_get .....	93
Table 3-94. Function can_flag_get .....	93
Table 3-95. Function can_flag_clear .....	94
Table 3-96. Function can_interrupt_enable .....	95
Table 3-97. Function can_interrupt_disable .....	96
Table 3-98. Function can_interrupt_flag_get.....	96
Table 3-99. Function can_interrupt_flag_clear .....	97
Table 3-100. Function can_fifo_overrun_flag_get .....	98
Table 3-101. CAU Registers .....	99
Table 3-102. CAU firmware function .....	99
Table 3-103. Structure cau_key_parameter_struct.....	100
Table 3-104. Structure cau_parameter_struct .....	100
Table 3-105. Function cau_deinit.....	100
Table 3-106. Function cau_struct_para_init .....	101
Table 3-107. Function cau_key_struct_para_init .....	101
Table 3-108. Function cau_enable.....	102
Table 3-109. Function cau_disable.....	102
Table 3-110. Function cau_dma_enable .....	103
Table 3-111. Function cau_dma_disable .....	103
Table 3-112. Function cau_init .....	104
Table 3-113. Function cau_aes_keysize_config.....	105
Table 3-114. Function cau_key_init .....	105
Table 3-115. Function cau_fifo_flush .....	106
Table 3-116. Function cau_enable_state_get .....	106
Table 3-117. Function cau_data_write .....	107
Table 3-118. Function cau_data_read .....	107
Table 3-119. Function cau_aes_ecb .....	108
Table 3-120. Function cau_flag_get .....	109
Table 3-121. Function cau_interrupt_enable .....	110
Table 3-122. Function cau_interrupt_disable .....	110
Table 3-123. Function cau_interrupt_flag_get.....	111
Table 3-124. CMP registers .....	111
Table 3-125. CMP firmware function .....	112
Table 3-126. Enum cmp_enum .....	112
Table 3-127. Function cmp_deinit .....	112

Table 3-128. Function cmp_mode_init.....	113
Table 3-129. Function cmp_output_init.....	114
Table 3-130. Function cmp_blanking_init.....	115
Table 3-131. Function cmp_digital_filter_init.....	115
Table 3-132. Function cmp_enable.....	116
Table 3-133. Function cmp_disable.....	117
Table 3-134. Function cmp_lock_enable.....	117
Table 3-135. Function cmp_voltage_scaler_enable.....	118
Table 3-136. Function cmp_voltage_scaler_disable.....	118
Table 3-137. Function cmp_scaler_bridge_enable.....	119
Table 3-138. Function cmp_scaler_bridge_disable.....	119
Table 3-139. Function cmp_output_level_get.....	120
Table 3-140. Function cmp_flag_get.....	120
Table 3-141. Function cmp_flag_clear.....	121
Table 3-142. Function cmp_interrupt_enable.....	122
Table 3-143. Function cmp_interrupt_disable.....	122
Table 3-144. Function cmp_interrupt_flag_get.....	123
Table 3-145. Function cmp_interrupt_flag_clear.....	123
Table 3-146. CRC Registers.....	124
Table 3-147. CRC firmware function.....	124
Table 3-148. Function crc_deinit.....	125
Table 3-149. Function crc_init_data_register_write.....	125
Table 3-150. Function crc_data_register_read.....	126
Table 3-151. Function crc_free_data_register_read.....	126
Table 3-152. Function crc_free_data_register_write.....	127
Table 3-153. Function crc_reverse_output_data_disable.....	127
Table 3-154. Function crc_reverse_output_data_enable.....	128
Table 3-155. Function crc_input_data_reverse_config.....	128
Table 3-156. Function crc_data_register_reset.....	129
Table 3-157. Function crc_polynomial_size_set.....	129
Table 3-158. Function crc_polynomial_set.....	130
Table 3-159. Function crc_single_data_calculate.....	130
Table 3-160. Function crc_block_data_calculate.....	131
Table 3-161. CTC Registers.....	132
Table 3-162. CTC firmware function.....	132
Table 3-163. Function ctc_deinit.....	133
Table 3-164. Function ctc_counter_enable.....	133
Table 3-165. Function ctc_counter_disable.....	134
Table 3-166. Function ctc_irc48m_trim_value_config.....	134
Table 3-167. Function ctc_software_refsource_pulse_generate.....	135
Table 3-168. Function ctc_hardware_trim_mode_config.....	135
Table 3-169. Function ctc_refsource_polarity_config.....	136
Table 3-170. Function ctc_refsource_signal_select.....	136
Table 3-171. Function ctc_refsource_prescaler_config.....	137

Table 3-172. Function <code>ctc_clock_limit_value_config</code> .....	138
Table 3-173. Function <code>ctc_counter_reload_value_config</code> .....	138
Table 3-174. Function <code>ctc_counter_capture_value_read</code> .....	139
Table 3-175. Function <code>ctc_counter_direction_read</code> .....	139
Table 3-176. Function <code>ctc_counter_reload_value_read</code> .....	140
Table 3-177. Function <code>ctc_irc48m_trim_value_read</code> .....	140
Table 3-178. Function <code>ctc_interrupt_enable</code> .....	141
Table 3-179. Function <code>ctc_interrupt_disable</code> .....	141
Table 3-180. Function <code>ctc_interrupt_flag_get</code> .....	142
Table 3-181. Function <code>ctc_interrupt_flag_clear</code> .....	143
Table 3-182. Function <code>ctc_flag_get</code> .....	143
Table 3-183. Function <code>ctc_flag_clear</code> .....	144
Table 3-184. DAC Registers .....	145
Table 3-185. DAC firmware functions .....	145
Table 3-186. Function <code>dac_deinit</code> .....	146
Table 3-187. Function <code>dac_enable</code> .....	147
Table 3-188. Function <code>dac_disable</code> .....	147
Table 3-189. Function <code>dac_dma_enable</code> .....	148
Table 3-190. Function <code>dac_dma_disable</code> .....	148
Table 3-191. Function <code>dac_dma_disable</code> .....	149
Table 3-192. Function <code>dac_output_buffer_enable</code> .....	150
Table 3-193. Function <code>dac_output_buffer_disable</code> .....	150
Table 3-194. Function <code>dac_output_value_get</code> .....	151
Table 3-195. Function <code>dac_data_set</code> .....	151
Table 3-196. Function <code>dac_trigger_enable</code> .....	152
Table 3-197. Function <code>dac_trigger_disable</code> .....	153
Table 3-198. Function <code>dac_trigger_source_config</code> .....	153
Table 3-199. Function <code>dac_software_trigger_enable</code> .....	154
Table 3-200. Function <code>dac_wave_mode_config</code> .....	155
Table 3-201. Function <code>dac_lfsr_noise_config</code> .....	155
Table 3-202. Function <code>dac_triangle_noise_config</code> .....	156
Table 3-203. Function <code>dac_output_connect_to_pin_enable</code> .....	157
Table 3-204. Function <code>dac_output_connect_to_pin_disable</code> .....	157
Table 3-205. Function <code>dac_flag_get</code> .....	158
Table 3-206. Function <code>dac_flag_clear</code> .....	158
Table 3-207. Function <code>dac_interrupt_enable</code> .....	159
Table 3-208. Function <code>dac_interrupt_disable</code> .....	160
Table 3-209. Function <code>dac_interrupt_flag_get</code> .....	160
Table 3-210. Function <code>dac_interrupt_flag_clear</code> .....	161
Table 3-211. DBG Registers .....	162
Table 3-212. DBG firmware function .....	162
Table 3-213. Enum <code>dbg_periph_enum</code> .....	162
Table 3-214. Function <code>dbg_deinit</code> .....	163
Table 3-215. Function <code>dbg_id_get</code> .....	163

Table 3-216. Function dbg_low_power_enable.....	164
Table 3-217. Function dbg_low_power_disable.....	164
Table 3-218. Function dbg_periph_enable.....	165
Table 3-219. Function dbg_periph_disable.....	165
Table 3-220. Function dbg_trace_pin_enable .....	166
Table 3-221. Function dbg_trace_pin_disable .....	167
Table 3-222. Function dbg_trace_pin_mode_set.....	167
Table 3-223. DMA Registers.....	168
Table 3-224. DMAMUX Registers .....	168
Table 3-225. DMA firmware function .....	169
Table 3-226. DMAMUX firmware function.....	169
Table 3-227. Structure dma_parameter_struct.....	170
Table 3-228. Structure dmamux_sync_parameter_struct .....	171
Table 3-229. Structure dmamux_gen_parameter_struct .....	171
Table 3-230. Enum dma_channel_enum .....	171
Table 3-231. Enum dmamux_multiplexer_channel_enum .....	171
Table 3-232. Enum dmamux_generator_channel_enum .....	172
Table 3-233. Enum dmamux_interrupt_enum .....	172
Table 3-234. Enum dmamux_flag_enum .....	173
Table 3-235. Enum dmamux_interrupt_flag_enum.....	174
Table 3-236. Function dma_deinit .....	175
Table 3-237. Function dma_struct_para_init .....	175
Table 3-238. Function dma_init.....	176
Table 3-239. Function dma_circulation_enable .....	177
Table 3-240. Function dma_circulation_disable .....	177
Table 3-241. Function dma_memory_to_memory_enable .....	178
Table 3-242. Function dma_memory_to_memory_disable .....	178
Table 3-243. Function dma_channel_enable .....	179
Table 3-244. Function dma_channel_disable .....	180
Table 3-245. Function dma_periph_address_config.....	180
Table 3-246. Function dma_memory_address_config.....	181
Table 3-247. Function dma_transfer_number_config.....	181
Table 3-248. Function dma_transfer_number_get.....	182
Table 3-249. Function dma_priority_config .....	183
Table 3-250. Function dma_memory_width_config .....	183
Table 3-251. Function dma_periph_width_config .....	184
Table 3-252. Function dma_memory_increase_enable .....	185
Table 3-253. Function dma_memory_increase_disable .....	185
Table 3-254. Function dma_periph_increase_enable .....	186
Table 3-255. Function dma_periph_increase_disable .....	187
Table 3-256. Function dma_transfer_direction_config.....	187
Table 3-257. Function dma_flag_get .....	188
Table 3-258. Function dma_flag_clear .....	189
Table 3-259. Function dma_interrupt_enable.....	189

Table 3-260. Function dma_interrupt_disable .....	190
Table 3-261. Function dma_interrupt_flag_get .....	191
Table 3-262. Function dma_interrupt_flag_clear .....	191
Table 3-263. Function dmamux_sync_struct_para_init.....	192
Table 3-264. Function dmamux_synchronization_init.....	193
Table 3-265. Function dmamux_synchronization_enable.....	194
Table 3-266. Function dmamux_synchronization_disable.....	194
Table 3-267. Function dmamux_event_generation_enable .....	195
Table 3-268. Function dmamux_event_generation_disable .....	195
Table 3-269. Function dmamux_gen_struct_para_init .....	196
Table 3-270. Function dmamux_request_generator_init.....	196
Table 3-271. Function dmamux_request_generator_channel_enable .....	197
Table 3-272. Function dmamux_request_generator_channel_disable .....	198
Table 3-273. Function dmamux_synchronization_polarity_config .....	198
Table 3-274. Function dmamux_request_forward_number_config.....	199
Table 3-275. Function dmamux_sync_id_config .....	200
Table 3-276. Function dmamux_request_id_config .....	201
Table 3-277. Function dmamux_trigger_polarity_config .....	208
Table 3-278. Function dmamux_request_generate_number_config.....	209
Table 3-279. Function dmamux_trigger_id_config.....	209
Table 3-280. Function dmamux_flag_get .....	211
Table 3-281. Function dmamux_flag_clear .....	212
Table 3-282. Function dmamux_interrupt_enable .....	212
Table 3-283. Function dmamux_interrupt_disable .....	213
Table 3-284. Function dmamux_interrupt_flag_get .....	213
Table 3-285. Function dmamux_interrupt_flag_clear .....	214
Table 3-286. EXMC Registers .....	215
Table 3-287. EXMC firmware function.....	215
Table 3-288. Structure exmc_norsram_timing_parameter_struct .....	216
Table 3-289. Structure exmc_norsram_parameter_struct.....	216
Table 3-290. Structure exmc_nand_timing_parameter_struct .....	217
Table 3-291. Structure exmc_nand_parameter_struct.....	217
Table 3-292. Function exmc_norsram_deinit .....	218
Table 3-293. Function exmc_norsram_struct_para_init.....	218
Table 3-294. Function exmc_norsram_init.....	219
Table 3-295. Function exmc_norsram_enable .....	220
Table 3-296. Function exmc_norsram_disable .....	221
Table 3-297. Function exmc_nand_deinit .....	222
Table 3-298. Function exmc_nand_struct_para_init .....	222
Table 3-299. Function exmc_nand_init.....	223
Table 3-300. Function exmc_nand_enable .....	224
Table 3-301. Function exmc_nand_disable .....	225
Table 3-302. Function exmc_norsram_page_size_config .....	225
Table 3-303. Function exmc_nand_ecc_enable .....	226



Table 3-304. Function <code>exmc_nand_ecc_disable</code> .....	227
Table 3-305. Function <code>exmc_ecc_get</code> .....	227
Table 3-306. Function <code>exmc_flag_get</code> .....	228
Table 3-307. EXTI Registers .....	229
Table 3-308. EXTI firmware function .....	229
Table 3-309. Enum <code>exti_line_enum</code> .....	229
Table 3-310. Enum <code>exti_mode_enum</code> .....	230
Table 3-311. Enum <code>exti_trig_type_enum</code> .....	230
Table 3-312. Function <code>exti_deinit</code> .....	230
Table 3-313. Function <code>exti_init</code> .....	231
Table 3-314. Function <code>exti_interrupt_enable</code> .....	231
Table 3-315. Function <code>exti_interrupt_disable</code> .....	232
Table 3-316. Function <code>exti_event_enable</code> .....	232
Table 3-317. Function <code>exti_event_disable</code> .....	233
Table 3-318. Function <code>exti_software_interrupt_enable</code> .....	233
Table 3-319. Function <code>exti_software_interrupt_disable</code> .....	234
Table 3-320. Function <code>exti_flag_get</code> .....	234
Table 3-321. Function <code>exti_flag_clear</code> .....	235
Table 3-322. Function <code>exti_interrupt_flag_get</code> .....	235
Table 3-323. Function <code>exti_interrupt_flag_clear</code> .....	236
Table 3-324. FMC Registers .....	236
Table 3-325. FMC firmware function .....	237
Table 3-326. Enum <code>fmc_state_enum</code> .....	238
Table 3-327. Enum <code>fmc_interrupt_enum</code> .....	239
Table 3-328. Enum <code>fmc_flag_enum</code> .....	239
Table 3-329. Enum <code>fmc_interrupt_flag_enum</code> .....	239
Table 3-330. Function <code>fmc_unlock</code> .....	239
Table 3-331. Function <code>fmc_bank0_unlock</code> .....	240
Table 3-332. Function <code>fmc_bank1_unlock</code> .....	240
Table 3-333. Function <code>fmc_lock</code> .....	241
Table 3-334. Function <code>fmc_bank0_lock</code> .....	241
Table 3-335. Function <code>fmc_bank1_lock</code> .....	242
Table 3-336. Function <code>fmc_page_erase</code> .....	242
Table 3-337. Function <code>fmc_mass_erase</code> .....	243
Table 3-338. Function <code>fmc_bank0_erase</code> .....	243
Table 3-339. Function <code>fmc_bank1_erase</code> .....	244
Table 3-340. Function <code>fmc_word_program</code> .....	244
Table 3-341. Function <code>fmc_halfword_program</code> .....	245
Table 3-342. Function <code>fmc_otpw_word_program</code> .....	245
Table 3-343. Function <code>fmc_otpw_half_word_program</code> .....	246
Table 3-344. Function <code>fmc_otpw_byte_program</code> .....	247
Table 3-345. Function <code>fmc_nwa_enable</code> .....	247
Table 3-346. Function <code>fmc_nwa_disable</code> .....	248
Table 3-347. Function <code>otp1_read_disable</code> .....	248

Table 3-348. Function otp2_rlock_enable .....	249
Table 3-349. Function fmc_deep_power_down_enable .....	249
Table 3-350. Function fmc_deep_power_down_disable .....	250
Table 3-351. Function ob_unlock .....	250
Table 3-352. Function ob_lock .....	250
Table 3-353. Function ob_start .....	251
Table 3-354. Function ob_erase .....	251
Table 3-355. Function ob_write_protection_enable .....	252
Table 3-356. Function ob_write_protection_disable .....	252
Table 3-357. Function ob_security_protection_config .....	253
Table 3-358. Function ob_user_write .....	254
Table 3-359. Function ob_sram_init_config .....	254
Table 3-360. Function ob_sram_ecc_config .....	255
Table 3-361. Function ob_nwa_clock_config .....	255
Table 3-362. Function ob_data_write .....	256
Table 3-363. Function ob_sram_init_get .....	256
Table 3-364. Function ob_sram_ecc_get .....	257
Table 3-365. Function ob_nwa_clock_get .....	257
Table 3-366. Function ob_data_get .....	258
Table 3-367. Function ob_write_protection_get .....	259
Table 3-368. Function ob_spc_get .....	259
Table 3-369. Function fmc_interrupt_enable .....	260
Table 3-370. Function fmc_interrupt_disable .....	260
Table 3-371. Function fmc_interrupt_flag_get .....	261
Table 3-372. Function fmc_interrupt_flag_clear .....	261
Table 3-373. Function fmc_flag_get .....	262
Table 3-374. Function fmc_flag_clear .....	263
Table 3-375. Function fmc_bank0_state_get .....	263
Table 3-376. Function fmc_bank1_state_get .....	264
Table 3-377. Function fmc_bank0_ready_wait .....	264
Table 3-378. Function fmc_bank1_ready_wait .....	265
Table 3-379. FWDGT Registers .....	266
Table 3-380. FWDGT firmware function .....	266
Table 3-381. Function fwdgt_write_enable .....	266
Table 3-382. Function fwdgt_write_disable .....	267
Table 3-383. Function fwdgt_enable .....	267
Table 3-384. Function fwdgt_prescaler_value_config .....	268
Table 3-385. Function fwdgt_reload_value_config .....	269
Table 3-386. Function fwdgt_counter_reload .....	269
Table 3-387. Function fwdgt_config .....	269
Table 3-388. Function fwdgt_flag_get fwdgt_write_disable .....	270
Table 3-389. GPIO Registers .....	271
Table 3-390. GPIO firmware function .....	272
Table 3-391. Function gpio_deinit .....	272

Table 3-392. Function gpio_afio_deinit .....	273
Table 3-393. Function gpio_mode_set.....	273
Table 3-394. Function gpio_output_options_set .....	274
Table 3-395. Function gpio_bit_set .....	275
Table 3-396. Function gpio_bit_reset .....	276
Table 3-397. Function gpio_bit_write.....	276
Table 3-398. Function gpio_port_write .....	277
Table 3-399. Function gpio_input_bit_get.....	278
Table 3-400. Function gpio_input_port_get.....	278
Table 3-401. Function gpio_output_bit_get .....	279
Table 3-402. Function gpio_output_port_get .....	279
Table 3-403. Function gpio_af_set .....	280
Table 3-404. Function gpio_exti_source_select .....	281
Table 3-405. Function gpio_pin_lock .....	281
Table 3-406. HAU Registers .....	282
Table 3-407. HAU firmware function .....	283
Table 3-408. Structure hau_digest_parameter_struct .....	283
Table 3-409. Function hau_deinit .....	283
Table 3-410. Function hau_init.....	284
Table 3-411. Function hau_reset.....	284
Table 3-412. Function hau_last_word_validbits_num_config .....	285
Table 3-413. Function hau_data_write .....	285
Table 3-414. Function hau_infifo_words_num_get .....	286
Table 3-415. Function hau_digest_read .....	286
Table 3-416. Function hau_digest_calculation_enable .....	287
Table 3-417. Function hau_multiple_single_dma_config.....	287
Table 3-418. Function hau_dma_enable.....	288
Table 3-419. Function hau_dma_disable.....	288
Table 3-420. Function hau_hash_sha_256.....	289
Table 3-421. Function hau_flag_get.....	289
Table 3-422. Function hau_flag_clear .....	290
Table 3-423. Function hau_interrupt_enable .....	291
Table 3-424. Function hau_interrupt_disable.....	291
Table 3-425. Function hau_interrupt_flag_get .....	292
Table 3-426. Function hau_interrupt_flag_clear .....	292
Table 3-427. I2C Registers .....	293
Table 3-428. I2C firmware function.....	294
Table 3-429. i2c_interrupt_flag_enum .....	295
Table 3-430. Function i2c_deinit .....	296
Table 3-431. Function i2c_timing_config .....	296
Table 3-432. Function i2c_digital_noise_filter_config .....	297
Table 3-433. Function i2c_analog_noise_filter_enable .....	298
Table 3-434. Function i2c_analog_noise_filter_disable .....	298
Table 3-435. Function i2c_master_clock_config .....	299

Table 3-436. Function i2c_master_addressing .....	299
Table 3-437. Function i2c_address10_header_enable .....	300
Table 3-438. Function i2c_address10_header_disable .....	301
Table 3-439. Function i2c_address10_enable .....	301
Table 3-440. Function i2c_address10_disable .....	302
Table 3-441. Function i2c_automatic_end_enable .....	302
Table 3-442. Function i2c_automatic_end_disable .....	303
Table 3-443. Function i2c_slave_response_to_gcall_enable .....	303
Table 3-444. Function i2c_slave_response_to_gcall_disable .....	304
Table 3-445. Function i2c_stretch_scl_low_enable .....	304
Table 3-446. Function i2c_stretch_scl_low_disable .....	305
Table 3-447. Function i2c_address_config .....	305
Table 3-448. Function i2c_address_bit_compare_config .....	306
Table 3-449. Function i2c_address_disable .....	307
Table 3-450. Function i2c_second_address_config .....	307
Table 3-451. Function i2c_second_address_disable .....	308
Table 3-452. Function i2c_received_address_get .....	309
Table 3-453. Function i2c_slave_byte_control_enable .....	309
Table 3-454. Function i2c_slave_byte_control_disable .....	310
Table 3-455. Function i2c_nack_enable .....	310
Table 3-456. Function i2c_enable .....	311
Table 3-457. Function i2c_disable .....	311
Table 3-458. Function i2c_start_on_bus .....	312
Table 3-459. Function i2c_stop_on_bus .....	312
Table 3-460. Function i2c_data_transmit .....	313
Table 3-461. Function i2c_data_receive .....	313
Table 3-462. Function i2c_reload_enable .....	314
Table 3-463. Function i2c_reload_disable .....	314
Table 3-464. Function i2c_transfer_byte_number_config .....	315
Table 3-465. Function i2c_dma_enable .....	315
Table 3-466. Function i2c_dma_disable .....	316
Table 3-467. Function i2c_pec_transfer .....	317
Table 3-468. Function i2c_pec_enable .....	317
Table 3-469. Function i2c_pec_disable .....	318
Table 3-470. Function i2c_pec_value_get .....	318
Table 3-471. Function i2c_smbus_alert_enable .....	319
Table 3-472. Function i2c_smbus_alert_disable .....	319
Table 3-473. Function i2c_smbus_default_addr_enable .....	320
Table 3-474. Function i2c_smbus_default_addr_disable .....	320
Table 3-475. Function i2c_smbus_host_addr_enable .....	321
Table 3-476. Function i2c_smbus_host_addr_disable .....	321
Table 3-477. Function i2c_extended_clock_timeout_enable .....	322
Table 3-478. Function i2c_extended_clock_timeout_disable .....	322
Table 3-479. Function i2c_clock_timeout_enable .....	323

Table 3-480. Function i2c_clock_timeout_disable.....	323
Table 3-481. Function i2c_bus_timeout_b_config.....	324
Table 3-482. Function i2c_bus_timeout_a_config.....	324
Table 3-483. Function i2c_idle_clock_timeout_config.....	325
Table 3-484. Function i2c_flag_get.....	325
Table 3-485. Function i2c_flag_clear.....	326
Table 3-486. Function i2c_interrupt_enable.....	327
Table 3-487. Function i2c_interrupt_disable .....	328
Table 3-488. Function i2c_interrupt_flag_get.....	328
Table 3-489. Function i2c_interrupt_flag_clear.....	330
Table 3-490. NVIC Registers .....	331
Table 3-491. SysTick Registers .....	332
Table 3-492. IRQn_Type.....	332
Table 3-493. MISC firmware function .....	334
Table 3-494. Function nvic_priority_group_set .....	334
Table 3-495. Function nvic_irq_enable.....	335
Table 3-496. Function nvic_irq_disable.....	335
Table 3-497. Function nvic_system_reset .....	336
Table 3-498. Function nvic_vector_table_set.....	336
Table 3-499. Function system_lowpower_set .....	337
Table 3-500. Function system_lowpower_reset.....	338
Table 3-501. Function systick_clksource_set .....	338
Table 3-502. PMU Registers.....	339
Table 3-503. PMU firmware function .....	339
Table 3-504. Function pmu_deinit .....	340
Table 3-505. Function pmu_lock.....	341
Table 3-506. Function pmu_unlock .....	342
Table 3-507. Function pmu_lvd_select.....	342
Table 3-508. Function pmu_pdrvs_select .....	343
Table 3-509. Function pmu_ldo_output_select.....	344
Table 3-510. Function pmu_lvd_enable.....	345
Table 3-511. Function pmu_lvd_disable.....	345
Table 3-512. Function pmu_lowdriver_mode_enable .....	346
Table 3-513. Function pmu_lowdriver_mode_disable .....	346
Table 3-514. Function pmu_lowpower_driver_config.....	347
Table 3-515. Function pmu_normalpower_driver_config .....	348
Table 3-516. Function pmu_to_sleepmode.....	348
Table 3-517. Function pmu_to_deepsleepmode .....	349
Table 3-518. Function pmu_to_standbymode .....	350
Table 3-519. Function pmu_wakeup_pin_enable .....	351
Table 3-520. Function pmu_wakeup_pin_disable .....	351
Table 3-521. Function pmu_backup_write_enable.....	352
Table 3-522. Function pmu_backup_write_disable .....	352
Table 3-523. Function pmu_vavd_enable.....	353

Table 3-524. Function pmu_vavd_disable .....	353
Table 3-525. Function pmu_vovd_enable .....	354
Table 3-526. Function pmu_vovd_disable .....	354
Table 3-527. Function pmu_vuvd_enable .....	355
Table 3-528. Function pmu_vuvd_disable .....	355
Table 3-529. Function pmu_avd_select.....	356
Table 3-530. Function pmu_vovd_enable .....	356
Table 3-531. Function pmu_vuvd_select.....	357
Table 3-532. Function pmu_vovdo_dnf_enable .....	358
Table 3-533. Function pmu_vuvdo_dnf_select .....	358
Table 3-534. Function pmu_flag_get.....	358
Table 3-535. Function pmu_flag_clear.....	359
Table 3-536. RCU Registers .....	360
Table 3-537. RCU firmware function .....	361
Table 3-538. Enum rcu_periph_enum .....	363
Table 3-539. Enum rcu_periph_sleep_enum .....	364
Table 3-540. Enum rcu_periph_reset_enum.....	364
Table 3-541. Enum rcu_flag_enum.....	366
Table 3-542. Enum rcu_int_flag_enum .....	366
Table 3-543. Enum rcu_flag_clear_enum.....	367
Table 3-544. Enum rcu_int_flag_clear_enum .....	367
Table 3-545. Enum rcu_int_enum.....	368
Table 3-546. Enum rcu_osci_type_enum .....	368
Table 3-547. Enum rcu_clock_freq_enum.....	368
Table 3-548. Enum rcu_ckfm_enum.....	369
Table 3-549. Function rcu_deinit .....	369
Table 3-550. Function rcu_periph_clock_enable .....	369
Table 3-551. Function rcu_periph_clock_disable.....	371
Table 3-552. Function rcu_periph_clock_sleep_enable .....	372
Table 3-553. Function rcu_periph_clock_sleep_disable .....	372
Table 3-554. Function rcu_periph_reset_enable.....	373
Table 3-555. Function rcu_periph_reset_disable .....	374
Table 3-556. Function rcu_bkp_reset_enable .....	375
Table 3-557. Function rcu_bkp_reset_disable .....	376
Table 3-558. Function rcu_system_clock_source_config.....	376
Table 3-559. Function rcu_system_clock_source_get .....	377
Table 3-560. Function rcu_ahb_clock_config .....	377
Table 3-561. Function rcu_apb1_clock_config .....	378
Table 3-562. Function rcu_apb2_clock_config .....	378
Table 3-563. Function rcu_ckout_config.....	379
Table 3-564. Function rcu_pll0_config .....	380
Table 3-565. Function rcu_pll1_config .....	381
Table 3-566. Function rcu_prediv0_config .....	382
Table 3-567. Function rcu_prediv1_config .....	382

Table 3-568. Function rcu_adc_clock_config.....	383
Table 3-569. Function rcu_usb_clock_config .....	384
Table 3-570. Function rcu_rtc_clock_config .....	384
Table 3-571. Function rcu_i2s1_clock_config.....	385
Table 3-572. Function rcu_i2s2_clock_config.....	386
Table 3-573. Function rcu_ck48m_clock_config .....	386
Table 3-574. Function rcu_fmc_clock_config .....	387
Table 3-575. Function rcu_lxtal_drive_capability_config.....	388
Table 3-576. Function rcu_osci_stab_wait .....	388
Table 3-577. Function rcu_osci_on .....	389
Table 3-578. Function rcu_osci_off.....	389
Table 3-579. Function rcu_osci_bypass_mode_enable .....	390
Table 3-580. Function rcu_osci_bypass_mode_disable .....	391
Table 3-581. Function rcu_irc8m_adjust_value_set.....	391
Table 3-582. Function rcu_hxtal_clock_monitor_enable .....	392
Table 3-583. Function rcu_hxtal_clock_monitor_disable .....	392
Table 3-584. Function rcu_lxtal_clock_monitor_enable.....	393
Table 3-585. Function rcu_lxtal_clock_monitor_disable.....	393
Table 3-586. Function rcu_clock_freq_monitor_enable .....	394
Table 3-587. Function rcu_clock_freq_monitor_disable .....	394
Table 3-588. Function rcu_irc8m_freq_monitor_config .....	395
Table 3-589. Function rcu_hxtal_monitor_threshold_config.....	395
Table 3-590. Function rcu_pll0p_monitor_threshold_config.....	396
Table 3-591. Function rcu_pll1_monitor_threshold_config.....	396
Table 3-592. Function rcu_deepsleep_voltage_set.....	397
Table 3-593. Function rcu_deepsleep_switch_delay_set.....	398
Table 3-594. Function rcu_reg_lock.....	398
Table 3-595. Function rcu_reg_unlock .....	399
Table 3-596. Function rcu_pll_bandwidth_config .....	399
Table 3-597. Function rcu_clock_freq_get.....	400
Table 3-598. Function rcu_flag_get.....	401
Table 3-599. Function rcu_flag_clear.....	402
Table 3-600. Function rcu_all_reset_flag_clear .....	402
Table 3-601. Function rcu_interrupt_enable.....	403
Table 3-602. Function rcu_interrupt_disable.....	404
Table 3-603. Function rcu_interrupt_flag_get .....	404
Table 3-604. Function rcu_interrupt_flag_clear .....	405
Table 3-605. RTC Registers .....	406
Table 3-606. RTC firmware function.....	407
Table 3-607. Function rtc_configuration_mode_enter.....	407
Table 3-608. Function rtc_configuration_mode_exit .....	408
Table 3-609. Function rtc_lwoff_wait .....	408
Table 3-610. Function rtc_register_sync_wait .....	409
Table 3-611. Function rtc_counter_get.....	409



Table 3-612. Function rtc_counter_set .....	410
Table 3-613. Function rtc_prescaler_set .....	410
Table 3-614. Function rtc_alarm_config .....	411
Table 3-615. Function rtc_divider_get .....	412
Table 3-616. Function rtc_interrupt_enable .....	412
Table 3-617. Function rtc_interrupt_disable .....	413
Table 3-618. Function rtc_flag_get .....	413
Table 3-619. Function rtc_flag_clear .....	414
Table 3-620. SYSCFG Registers .....	415
Table 3-621. SYSCFG firmware function .....	416
Table 3-622. Enum timer_channel_input_enum .....	416
Table 3-623. Function syscfg_deinit .....	417
Table 3-624. Function syscfg_bootmode_select .....	417
Table 3-625. Function syscfg_i2c_fast_mode_plus_enable .....	418
Table 3-626. Function syscfg_i2c_fast_mode_plus_disable .....	418
Table 3-627. Function syscfg_lockup_enable .....	419
Table 3-628. Function syscfg_bus_timeout_enable .....	419
Table 3-629. Function syscfg_bus_timeout_disable .....	420
Table 3-630. Function syscfg_timer_input_source_select .....	421
Table 3-631. Function syscfg_sram_page_wp_enable .....	421
Table 3-632. Function syscfg_sram_ecc_error_bit_get .....	423
Table 3-633. Function syscfg_sram_ecc_error_addr_get .....	424
Table 3-634. Function syscfg_fpu_interrupt_enable .....	424
Table 3-635. Function syscfg_fpu_interrupt_disable .....	425
Table 3-636. syscfg_sram_ecc_interrupt_enable .....	426
Table 3-637. syscfg_sram_ecc_interrupt_disable .....	427
Table 3-638. syscfg_bus_timeout_flag_get .....	427
Table 3-639. syscfg_bus_timeout_flag_clear .....	428
Table 3-640. syscfg_sram_ecc_flag_get .....	428
Table 3-641. syscfg_sram_ecc_flag_clear .....	429
Table 3-642. SPI/I2S Registers .....	430
Table 3-643. SPI/I2S firmware function .....	430
Table 3-644. spi_parameter_struct .....	431
Table 3-645. spi_interrupt_flag_enum .....	432
Table 3-646. Function spi_i2s_deinit .....	432
Table 3-647. Function spi_struct_para_init .....	433
Table 3-648. Function spi_init .....	433
Table 3-649. Function spi_enable .....	434
Table 3-650. Function spi_disable .....	434
Table 3-651. Function i2s_init .....	435
Table 3-652. Function i2s_psc_config .....	436
Table 3-653. Function i2s_enable .....	437
Table 3-654. Function i2s_disable .....	438
Table 3-655. Function spi_nss_output_enable .....	438



Table 3-656. Function spi_nss_output_disable .....	439
Table 3-657. Function spi_nss_internal_high .....	439
Table 3-658. Function spi_nss_internal_low .....	440
Table 3-659. Function spi_dma_enable .....	440
Table 3-660. Function spi_dma_disable .....	441
Table 3-661. Function spi_i2s_data_frame_format_config .....	441
Table 3-662. Function spi_i2s_data_transmit.....	442
Table 3-663. Function spi_i2s_data_receive .....	443
Table 3-664. Function spi_bidirectional_transfer_config.....	443
Table 3-665. Function spi_crc_polynomial_set.....	444
Table 3-666. Function spi_crc_polynomial_get .....	444
Table 3-667. Function spi_crc_on.....	445
Table 3-668. Function spi_crc_off .....	445
Table 3-669. Function spi_crc_next .....	446
Table 3-670. Function spi_crc_get.....	446
Table 3-671. Function spi_crc_error_clear .....	447
Table 3-672. Function spi_ti_mode_enable .....	448
Table 3-673. Function spi_ti_mode_disable .....	448
Table 3-674. Function spi_nssp_mode_enable.....	449
Table 3-675. Function spi_nssp_mode_disable.....	449
Table 3-676. Function spi_quad_enable.....	450
Table 3-677. Function spi_quad_disable.....	450
Table 3-678. Function spi_quad_write_enable.....	451
Table 3-679. Function spi_quad_read_enable.....	451
Table 3-680. Function spi_i2s_flag_get .....	452
Table 3-681. Function spi_i2s_interrupt_enable.....	453
Table 3-682. Function spi_i2s_interrupt_disable.....	453
Table 3-683. Function spi_i2s_interrupt_flag_get .....	454
Table 3-684. TIMERx Registers .....	455
Table 3-685. TIMERx firmware function.....	456
Table 3-686. Structure timer_parameter_struct .....	460
Table 3-687. Structure timer_break_parameter_struct.....	460
Table 3-688. Structure timer_oc_parameter_struct.....	461
Table 3-689. Structure timer_omc_parameter_struct .....	461
Table 3-690. Structure timer_ic_parameter_struct.....	461
Table 3-691. Function timer_deinit.....	462
Table 3-692. Function timer_struct_para_init.....	462
Table 3-693. Function timer_init .....	463
Table 3-694. Function timer_enable .....	463
Table 3-695. Function timer_disable .....	464
Table 3-696. Function timer_auto_reload_shadow_enable .....	464
Table 3-697. Function timer_auto_reload_shadow_disable .....	465
Table 3-698. Function timer_update_event_enable .....	465
Table 3-699. Function timer_update_event_disable .....	466

Table 3-700. Function timer_counter_alignment .....	466
Table 3-701. Function timer_counter_up_direction .....	467
Table 3-702. Function timer_counter_down_direction .....	468
Table 3-703. Function timer_prescaler_config.....	468
Table 3-704. Function timer_repetition_value_config .....	469
Table 3-705. Function timer_runtime_repetition_value_read.....	469
Table 3-706. Function timer_autoreload_value_config .....	470
Table 3-707. Function timer_autoreload_value_read.....	471
Table 3-708. Function timer_counter_value_config.....	471
Table 3-709. Function timer_counter_read .....	472
Table 3-710. Function timer_prescaler_read .....	472
Table 3-711. Function timer_single_pulse_mode_config.....	473
Table 3-712. Function timer_update_source_config.....	474
Table 3-713. Function timer_dma_enable .....	474
Table 3-714. Function timer_dma_disable .....	475
Table 3-715. Function timer_channel_dma_request_source_select.....	476
Table 3-716. Function timer_dma_transfer_config.....	476
Table 3-717. Function timer_event_software_generate.....	479
Table 3-718. Function timer_break_struct_para_init .....	480
Table 3-719. Function timer_break_config .....	480
Table 3-720. Function timer_break_enable.....	481
Table 3-721. Function timer_break_disable.....	482
Table 3-722. Function timer_automatic_output_enable .....	482
Table 3-723. Function timer_automatic_output_disable .....	483
Table 3-724. Function timer_primary_output_config.....	483
Table 3-725. Function timer_channel_control_shadow_config .....	484
Table 3-726. Function timer_channel_control_shadow_update_config.....	485
Table 3-727. Function timer_channel_output_struct_para_init .....	485
Table 3-728. Function timer_channel_output_config .....	486
Table 3-729. Function timer_channel_output_mode_config .....	487
Table 3-730. Function timer_channel_output_pulse_value_config.....	488
Table 3-731. Function timer_channel_output_shadow_config .....	489
Table 3-732. Function timer_channel_output_compare_fast_config.....	490
Table 3-733. Function timer_channel_output_clear_config .....	491
Table 3-734. Function timer_channel_output_polarity_config.....	491
Table 3-735. Function timer_channel_complementary_output_polarity_config .....	492
Table 3-736. Function timer_channel_output_state_config .....	493
Table 3-737. Function timer_channel_complementary_output_state_config .....	494
Table 3-738. Function timer_channel_composite_asymmetric_pwm_level_config.....	495
Table 3-739. Function timer_channel_output_clear_invalid_selection .....	496
Table 3-740. Function timer_channel_input_struct_para_init.....	496
Table 3-741. Function timer_input_capture_config.....	497
Table 3-742. Function timer_channel_input_capture_prescaler_config .....	498
Table 3-743. Function timer_channel_capture_value_register_read .....	499

Table 3-744. Function timer_input_pwm_capture_config .....	499
Table 3-745. Function timer_hall_mode_config .....	500
Table 3-746. Function timer_multi_mode_channel_output_parameter_struct_init .....	501
Table 3-747. Function timer_multi_mode_channel_output_config .....	502
Table 3-748. Function timer_multi_mode_channel_mode_config .....	502
Table 3-749. Function timer_input_trigger_source_select .....	503
Table 3-750. Function timer_master_output0_trigger_source_select .....	504
Table 3-751. Function timer_slave_mode_select .....	505
Table 3-752. Function timer_master_slave_mode_config .....	506
Table 3-753. Function timer_external_trigger_config .....	507
Table 3-754. Function timer_quadrature_decoder_mode_config .....	508
Table 3-755. Function timer_non_quadrature_decoder_mode_config .....	509
Table 3-756. Function timer_internal_clock_config .....	510
Table 3-757. Function timer_internal_trigger_as_external_clock_config .....	511
Table 3-758. Function timer_external_trigger_as_external_clock_config .....	512
Table 3-759. Function timer_external_clock_mode0_config .....	513
Table 3-760. Function timer_external_clock_mode1_config .....	514
Table 3-761. Function timer_external_clock_mode1_disable .....	514
Table 3-762. Function timer_write_chxval_register_config .....	515
Table 3-763. Function timer_output_value_selection_config .....	516
Table 3-764. Function timer_commutation_control_shadow_register_config .....	516
Table 3-765. Function timer_output_match_pulse_select .....	517
Table 3-766. Function timer_channel_composite_pwm_mode_config .....	518
Table 3-767. Function timer_channel_composite_pwm_mode_output_pulse_value_config .....	519
Table 3-768. Function timer_channel_asymmetric_pwm_pulse_config .....	519
Table 3-769. Function timer_channel_additional_compare_value_config .....	520
Table 3-770. Function timer_channel_additional_output_update_select .....	521
Table 3-771. Function timer_channel_additional_output_shadow_config .....	522
Table 3-772. Function timer_channel_additional_compare_value_read .....	523
Table 3-773. Function timer_break_external_source_config .....	523
Table 3-774. Function timer_break_external_polarity_config .....	524
Table 3-775. Function timer_dead_time_falling_edge_config .....	525
Table 3-776. Function timer_dead_time_different_config .....	525
Table 3-777. Function timer_channel_break_control_config .....	526
Table 3-778. Function timer_channel_dead_time_config .....	527
Table 3-779. Function timer_channel_break_config .....	527
Table 3-780. Function timer_channel_break_external_status_config .....	528
Table 3-781. Function timer_channel_break_external_polarity_config .....	529
Table 3-782. Function timer_channel_primary_output_config .....	530
Table 3-783. Function timer_channel_break_period_config .....	531
Table 3-784. Function timer_watchdog_value_config .....	531
Table 3-785. Function timer_watchdog_value_read .....	532
Table 3-786. Function timer_decoder_disconnection_detection_config .....	532
Table 3-787. Function timer_decoder_jump_detection_config .....	533

Table 3-788. Function timer_counter_initial_register_config.....	533
Table 3-789. Function timer_counter_initial_config.....	534
Table 3-790. Function timer_synchronization_event_generate .....	535
Table 3-791. Function timer_flag_get .....	535
Table 3-792. Function timer_flag_clear .....	537
Table 3-793. Function timer_interrupt_enable .....	538
Table 3-794. Function timer_interrupt_disable .....	539
Table 3-795. Function timer_interrupt_flag_get.....	540
Table 3-796. Function timer_interrupt_flag_clear.....	542
Table 3-797 TRIGSEL Registers .....	543
Table 3-798. TRIGSEL firmware function .....	544
Table 3-799. Enum trigselsource_enum.....	544
Table 3-800. Enum trigsels_periph_enum.....	548
Table 3-801. Function trigsels_deinit.....	549
Table 3-802. Function trigsels_init .....	550
Table 3-803. Function trigsels_trigger_source_get.....	550
Table 3-804. Function trigsels_register_lock_set.....	551
Table 3-805. Function trigsels_register_lock_get.....	551
Table 3-806 TRNG Registers .....	552
Table 3-807. TRNG firmware function.....	552
Table 3-808. Enum trng_flag_enum .....	552
Table 3-809. Enum trng_int_flag_enum.....	553
Table 3-810. Function trng_deinit.....	553
Table 3-811. Function trng_enable .....	553
Table 3-812 Function trng_disable.....	554
Table 3-813 Function trng_get_true_random_data .....	554
Table 3-814 trng_flag_get.....	555
Table 3-815 trng_flag_get.....	555
Table 3-816 trng_interrupt_enable .....	556
Table 3-817 trng_interrupt_disable .....	556
Table 3-818 trng_interrupt_flag_get .....	557
Table 3-819 trng_interrupt_flag_clear.....	557
Table 3-820. USART Registers .....	558
Table 3-821. USART firmware function.....	558
Table 3-822. Enum usart_flag_enum.....	560
Table 3-823. Enum usart_interrupt_flag_enum.....	560
Table 3-824. Enum usart_interrupt_enum.....	561
Table 3-825. Enum usart_invert_enum .....	561
Table 3-826. Function usart_deinit.....	561
Table 3-827. Function usart_baudrate_set .....	562
Table 3-828. Function usart_parity_config .....	562
Table 3-829. Function usart_word_length_set.....	563
Table 3-830. Function usart_stop_bit_set.....	564
Table 3-831. Function usart_enable .....	564

Table 3-832. Function <code>usart_disable</code> .....	565
Table 3-833. Function <code>usart_transmit_config</code> .....	565
Table 3-834. Function <code>usart_receive_config</code> .....	566
Table 3-835. Function <code>usart_data_first_config</code> .....	567
Table 3-836. Function <code>usart_invert_config</code> .....	567
Table 3-837. Function <code>usart_oversample_config</code> .....	568
Table 3-838. Function <code>usart_sample_bit_config</code> .....	569
Table 3-839. Function <code>usart_receiver_timeout_enable</code> .....	569
Table 3-840. Function <code>usart_receiver_timeout_disable</code> .....	570
Table 3-841. Function <code>usart_receiver_timeout_threshold_config</code> .....	570
Table 3-842. Function <code>usart_data_transmit</code> .....	571
Table 3-843. Function <code>usart_data_receive</code> .....	572
Table 3-844. Function <code>usart_address_config</code> .....	572
Table 3-845. Function <code>usart_mute_mode_enable</code> .....	573
Table 3-846. Function <code>usart_mute_mode_disable</code> .....	573
Table 3-847. Function <code>usart_mute_mode_wakeup_config</code> .....	574
Table 3-848. Function <code>usart_lin_mode_enable</code> .....	574
Table 3-849. Function <code>usart_lin_mode_disable</code> .....	575
Table 3-850. Function <code>usart_lin_break_dection_length_config</code> .....	575
Table 3-851. Function <code>usart_send_break</code> .....	576
Table 3-852. Function <code>usart_halfduplex_enable</code> .....	577
Table 3-853. Function <code>usart_halfduplex_disable</code> .....	577
Table 3-854. Function <code>usart_synchronous_clock_enable</code> .....	578
Table 3-855. Function <code>usart_synchronous_clock_disable</code> .....	578
Table 3-856. Function <code>usart_synchronous_clock_config</code> .....	579
Table 3-857. Function <code>usart_guard_time_config</code> .....	579
Table 3-858. Function <code>usart_smartcard_mode_enable</code> .....	580
Table 3-859. Function <code>usart_smartcard_mode_disable</code> .....	581
Table 3-860. Function <code>usart_smartcard_mode_nack_enable</code> .....	581
Table 3-861. Function <code>usart_smartcard_mode_nack_disable</code> .....	582
Table 3-862. Function <code>usart_smartcard_autoretry_config</code> .....	582
Table 3-863. Function <code>usart_block_length_config</code> .....	583
Table 3-864. Function <code>usart_irda_mode_enable</code> .....	583
Table 3-865. Function <code>usart_irda_mode_disable</code> .....	584
Table 3-866. Function <code>usart_prescaler_config</code> .....	584
Table 3-867. Function <code>usart_irda_lowpower_config</code> .....	585
Table 3-868. Function <code>usart_hardware_flow_rts_config</code> .....	585
Table 3-869. Function <code>usart_hardware_flow_cts_config</code> .....	586
Table 3-870. Function <code>usart_break_frame_coherence_config</code> .....	587
Table 3-871. Function <code>usart_parity_check_coherence_config</code> .....	587
Table 3-872. Function <code>usart_hardware_flow_coherence_config</code> .....	588
Table 3-873. Function <code>usart_dma_receive_config</code> .....	589
Table 3-874. Function <code>usart_dma_transmit_config</code> .....	589
Table 3-875. Function <code>usart_flag_get</code> .....	590

Table 3-876. Function <code>usart_flag_clear</code> .....	590
Table 3-877. Function <code>usart_interrupt_enable</code> .....	591
Table 3-878. Function <code>usart_interrupt_disable</code> .....	592
Table 3-879. Function <code>usart_interrupt_flag_get</code> .....	592
Table 3-880. Function <code>usart_interrupt_flag_clear</code> .....	593
Table 3-881. WWDGT Registers .....	594
Table 3-882. WWDGT firmware function .....	594
Table 3-883. Function <code>wwdgt_deinit</code> .....	594
Table 3-884. Function <code>wwdgt_enable</code> .....	595
Table 3-885. Function <code>wwdgt_counter_update</code> .....	595
Table 3-886. Function <code>wwdgt_config</code> .....	596
Table 3-887. Function <code>wwdgt_flag_get</code> .....	597
Table 3-888. Function <code>wwdgt_flag_clear</code> .....	598
Table 3-889. Function <code>wwdgt_interrupt_enable</code> .....	599
Table 4-1. Revision history .....	600

## 1. Introduction

This manual introduces firmware library of GD32F50x 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 GD32F50x devices are involved in the package. The peripheral driving code and firmware examples on evaluation board are also included in firmware library. Users need not learn each peripherals in details and it's easy to apply a peripheral by using the firmware library. Using firmware library can greatly reduce programming time, thereby reducing development costs.

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

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

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

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

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

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

#### 1.1.1. Peripherals

Table 1-1. Peripherals

Peripherals	Descriptions
ADC	Analog-to-digital converter
BKP	Backup registers
CAN	Controller area network
CAU	Cryptographic Acceleration Unit
CMP	Comparator

Peripherals	Descriptions
CRC	CRC calculation unit
CTC	Clock trim controller
DAC	Digital-to-analog converter
DBG	Debug
DMA	Direct memory access controller
DMAMUX	DMA request multiplexer
EXMC	External memory controller
EXTI	Interrupt/event controller
FMC	Flash memory controller
FWDGT	Free watchdog timer
GPIO	General-purpose I/Os
HAU	Hash Acceleration Unit
I2C	Inter-integrated circuit interface
MISC	Nested Vectored Interrupt Controller
PMU	Power management unit
RCU	Reset and clock unit
RTC	Reset and clock unit
SPI/I2S	Secure digital input/output interface
SYSCFG	System configuration
TIMER	TIMER
TRIGSEL	Trigger selection controller
TRNG	True random number generator
USART	Universal synchronous / asynchronous receiver / transmitter
USBFS	Universal serial bus full-speed interface
WWDGT	Window watchdog timer

### 1.1.2. Naming rules

The firmware library naming rules are shown as below:

- The peripherals are shortened in XXX format, such as: ADC. More shorten information of peripherals refer to [Peripherals](#);
- The name of sourcefile and header file are started with “gd32f50x\_”, such as: gd32f50x\_adc.h;
- The constants used only in one file should be defined in the used file; the constants used in many files should be defined in corresponding header file. All the constants are written in uppcase of English letters;
- Registers are handled as constants. The naming of them are written in uppcase of English letters. In most cases, register names are shortened accord with the user manual;
- Variables are written in lowercase, when concluded by several words, underlines should be adapted among words;
- The naming of peripheral functions are started with the peripheral abbreviation added



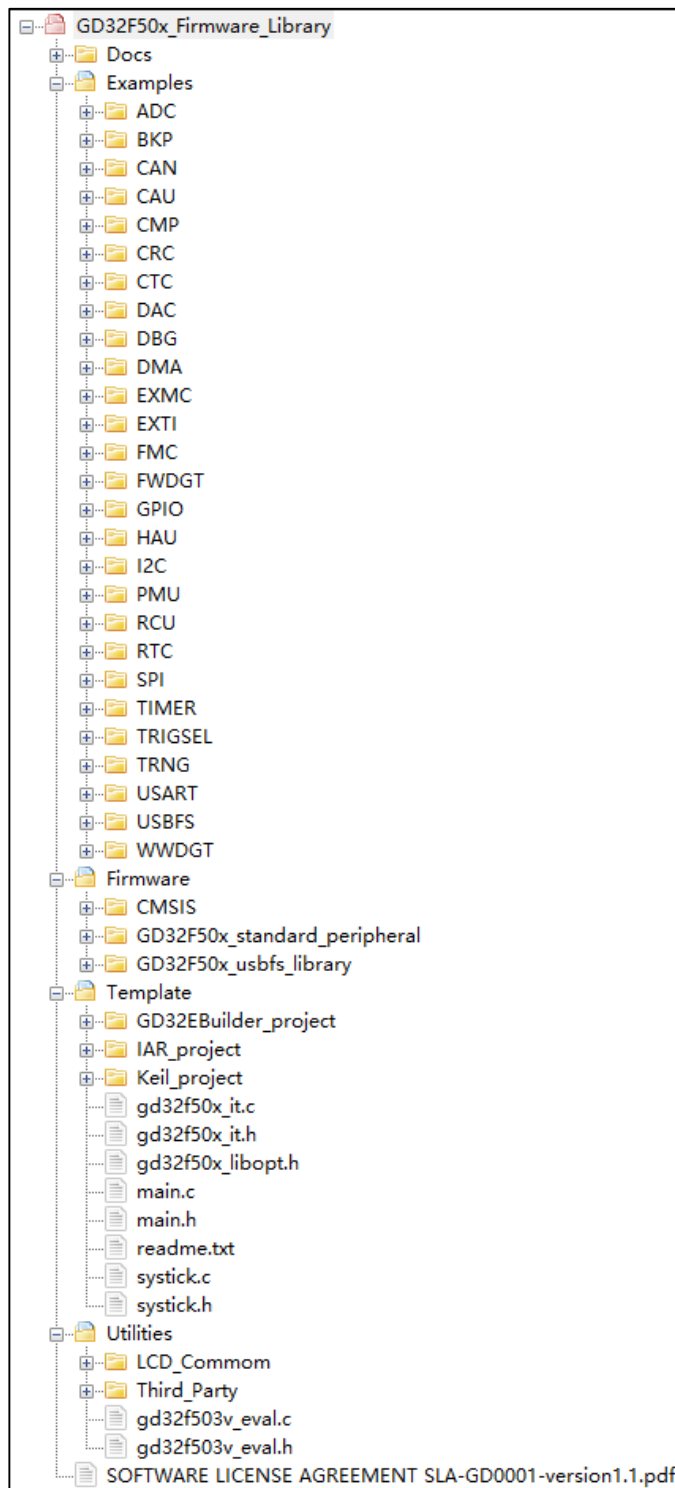
with an underline, when the function name is concluded by several words, underlines should be adapted among words, and all the peripheral functions are written in lower case.

## 2. Firmware Library Overview

### 2.1. File Structure of Firmware Library

GD32F50x\_Firmware\_Library, the file structure is shown as below:

**Figure 2-1. File structure of firmware library of GD32F50x**



### 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;
- `gd32f50x_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);
- `gd32f50x_it.c`: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- `gd32f50x_it.h`: the header file include all the prototypes of the interrupt service routines;
- `systick.c`: the source file include the precise time delay functions by using `systick`;
- `systick.h`: the header file include the prototype of the precise time delay functions by using `systick`;
- `main.c`: example code. Note: all the examples are not influenced by software IDEs.

### 2.1.2. Firmware Folder

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

- CMSIS subfolder includes the Cortex® M33 kernel support files, the startup file based on the Cortex® M33 kernel processor, the global header file of GD32F50x and system configuration file;
- GD32F50x\_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;

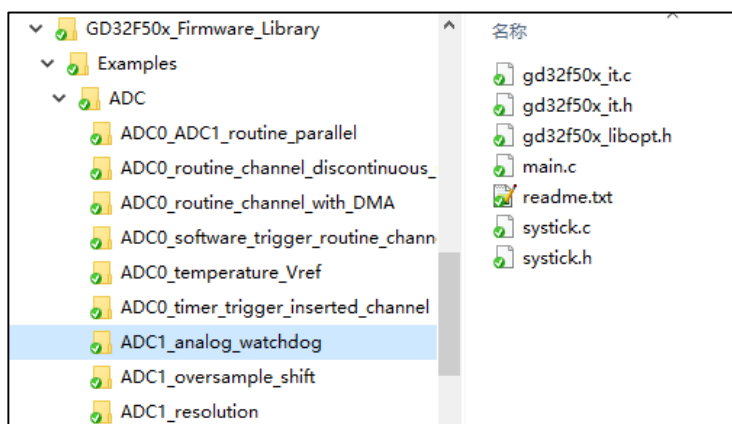
**Note:** All the codes accord with MISRA-C:2004 standard, and will not be influenced by different software IDEs.

### 2.1.3. Template Folder

Template folder includes a simple demo of how to use LED, how to print by USART and use key to control, (IAR\_project is run in IAR, and Keil\_project is run in Keil5, and GD32EBuilder\_project is run in GD32EBuilder). User can use the project template to compile the formware examples, the steps are shown as below:

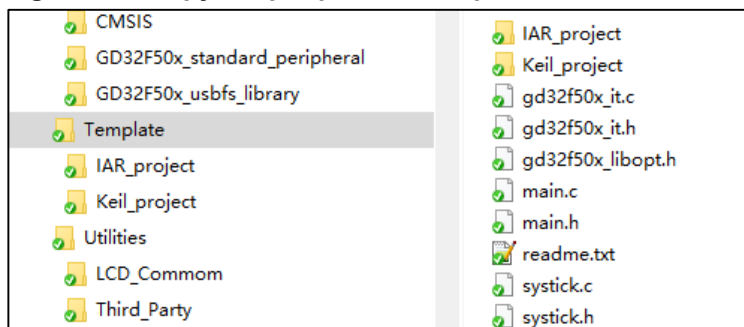
#### Select files

Open "Examples" folder, select the module to be tested, such as ADC, open "ADC" folder, select an example of ADC, such as "ADC1\_analog\_watchdog", shown as below:

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

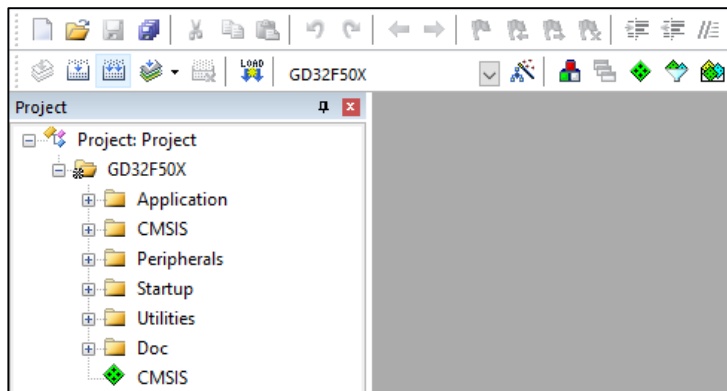
### Copy files

Open “Template” folder, keep the folders of ” IAR\_project”, ” Keil\_project” and “GD32EBuilder\_project”, and delete the other files, then copy all the files in “ADC1\_analog\_watchdog0” folder to the “Template” subfolder, shown as below:

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

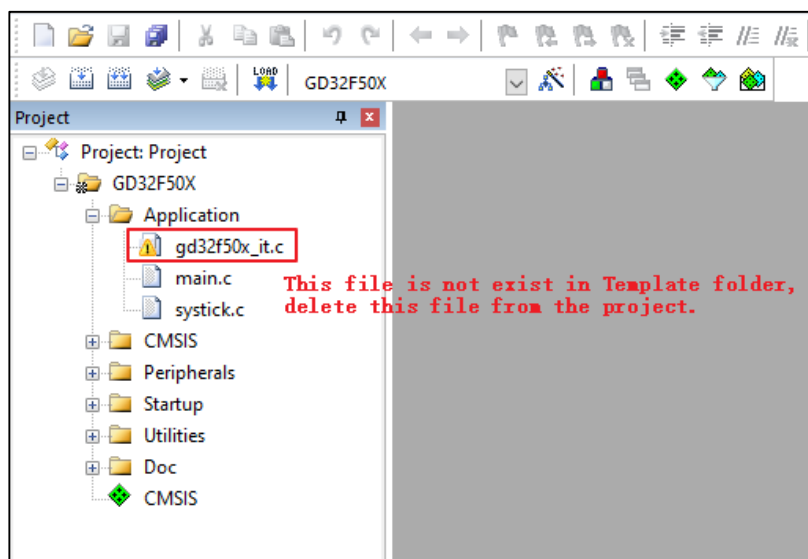
### Open a project

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

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

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

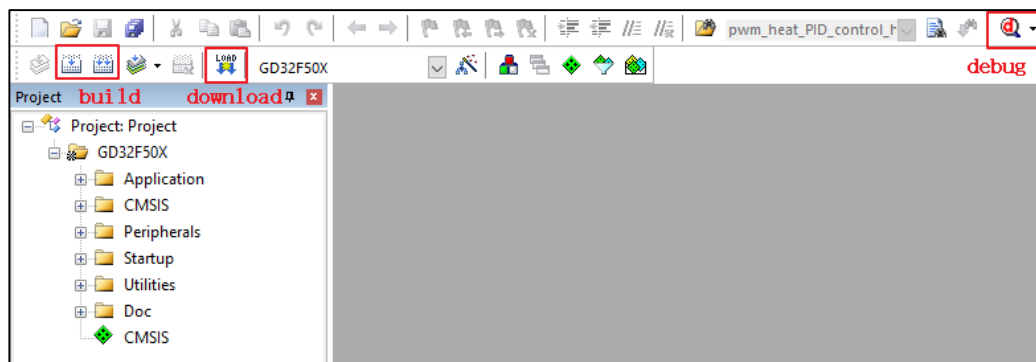
**Figure 2-5. Configure project files**



## Compile-Debug-Download

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

**Figure 2-6. Compile-debug-download**



### 2.1.4. Utilities Folder

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

- gd32f503v\_eval.h is related header file of the evaluation board about running the firmware examples;
- gd32f503v\_eval.c is related source file 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
gd32f50x_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.
gd32f50x_it.h	Header file, including all the prototypes of interrupt service routines.
gd32f50x_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.
gd32f50x_xxx.h	The header file of peripheral xxx, including functions about peripheral xxx, and the variables used for functions.
gd32f50x_xxx.c	The C source file for driving peripheral xxx.
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

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

#### 3.2.1. Descriptions of Peripheral registers

ADC registers are listed in the table shown as below:

**Table 3-2. ADC Registers**

<b>Registers</b>	<b>Descriptions</b>
ADC_STAT	ADC status register
ADC_CTL0	ADC control register 0
ADC_CTL1	ADC control register 1
ADC_SAMPT0	ADC sample time register 0
ADC_SAMPT1	ADC sample time register 1
ADC_IOFFx(x=0..3)	ADC inserted channel data offset register x
ADC_WD0HT	ADC watchdog 0 high threshold register
ADC_WD0LT	ADC watchdog 0 low threshold register

Registers	Descriptions
ADC_RSQ0	ADC routine sequence register 0
ADC_RSQ1	ADC routine sequence register 1
ADC_RSQ2	ADC routine sequence register 2
ADC_ISQ	ADC inserted sequence register
ADC_LDATAB(x=0..3)	ADC latch data register x
ADC_RDATA	ADC routine data register
ADC_IDATA	ADC inserted data register
ADC_LDCTL	ADC latch data control register
ADC_OVSAMPCTL	ADC 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 ADC
adc_enable	enable ADC interface
adc_disable	disable ADC interface
adc_dma_mode_enable	enable DMA request
adc_dma_mode_disable	disable DMA request
adc_discontinuous_mode_config	configure ADC discontinuous mode
adc_special_function_config	enable or disable ADC special function
adc_internal_channel_config	enable or disable ADC internal channels
adc_sync_mode_config	configure the ADC sync mode
adc_sync_routine_data_read	read ADC0 and ADC1 routine data in sync mode
adc_data_alignment_config	configure ADC data alignment
adc_channel_length_config	configure the channel length of routine sequence or inserted sequence
adc_routine_channel_config	configure ADC routine channel
adc_inserted_channel_config	configure ADC inserted channel
adc_inserted_channel_offset_config	configure ADC inserted channel offset
adc_external_trigger_config	enable ADC external trigger
adc_software_trigger_enable	enable ADC software trigger
adc_routine_data_read	read ADC routine sequence data register
adc_inserted_data_read	read ADC inserted sequence data register
adc_latch_data_read	read ADC latch data register
adc_latch_data_source_config	configure ADC latch data source
adc_watchdog0_single_channel_enable	configure ADC analog watchdog single channel
adc_watchdog0_sequence_channel_enable	enable ADC analog watchdog 0 sequence channel
adc_watchdog0_disable	disable ADC analog watchdog 0



Function name	Function description
adc_watchdog0_threshold_config	configure ADC analog watchdog 0 threshold
adc_resolution_config	configure ADC resolution
adc_oversample_mode_config	configure ADC oversample mode
adc_oversample_mode_enable	enable ADC oversample mode
adc_oversample_mode_disable	disable ADC oversample mode
adc_flag_get	get ADC flag
adc_flag_clear	clear ADC flag
adc_interrupt_enable	enable ADC interrupt
adc_interrupt_disable	disable ADC interrupt
adc_interrupt_flag_get	get ADC interrupt flag
adc_interrupt_flag_clear	clear ADC interrupt flag

## adc\_deinit

The description of adc\_deinit is shown as below:

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

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

Example:

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

## adc\_enable

The description of adc\_enable is shown as below:

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

Function name	adc_enable
Function prototype	void adc_enable(uint32_t adc_periph);
Function descriptions	enable ADC interface
Precondition	-
The called functions	-

Input parameter{in}	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0,1,2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_enable(ADC0);
```

### adc\_disable

The description of adc\_disable is shown as below:

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

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

Example:

```
/* disable ADC */
```

```
adc_disable(ADC0);
```

### adc\_dma\_mode\_enable

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

**Table 3-7. 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, uint8_t adc_sequence);
<b>Function descriptions</b>	enable DMA request
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0,1,2)</i>	ADC peripheral selection
Input parameter{in}	
<b>adc_sequence</b>	select the sequence
<i>ADC_ROUTINE_CHAN NEL</i>	routine sequence
<i>ADC_INSERTED_ CHANNEL</i>	inserted sequence
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_dma_mode_enable(ADC0, ADC_ROUTINE_CHANNEL);
```

### adc\_dma\_mode\_disable

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

**Table 3-8. 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, uint8_t adc_sequence);
<b>Function descriptions</b>	disable DMA request
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0,1,2)</i>	ADC peripheral selection
Input parameter{in}	
<b>adc_sequence</b>	select the sequence
<i>ADC_ROUTINE_CHAN NEL</i>	routine sequence
<i>ADC_INSERTED_ CHANNEL</i>	inserted sequence
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_dma_mode_disable(ADC0, ADC_ROUTINE_CHANNEL);
```

### adc\_discontinuous\_mode\_config

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

**Table 3-9. 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_sequence, uint8_t length);
<b>Function descriptions</b>	configure ADC discontinuous mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
ADCx(x=0,1,2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
ADC_ROUTINE_CHANNEL	routine sequence
ADC_INSERTED_CHANNEL	inserted sequence
ADC_CHANNEL_DISCON_DISABLE	disable discontinuous mode of routine and inserted sequence
<b>Input parameter{in}</b>	
<b>length</b>	number of conversions in discontinuous mode, the number can be 1..8 for routine channel, the number has no effect for inserted channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC0 routine sequence discontinuous mode */
```

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

### adc\_special\_function\_config

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

**Table 3-10. 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=0,1,2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>function</b>	the function to configure
<i>ADC_SCAN_MODE</i>	scan mode select
<i>ADC_INSERTED_CHANNEL_AUTO</i>	inserted sequence 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\_internal\_channel\_config

The description of `adc_internal_channel_config` is shown as below:

**Table 3-11. Function `adc_internal_channel_config`**

<b>Function name</b>	<code>adc_internal_channel_config</code>
<b>Function prototype</b>	<code>void adc_internal_channel_config(uint32_t internal_channel, ControlStatus newvalue);</code>
<b>Function descriptions</b>	enable or disable ADC internal channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>internal_channel</b>	the internal channels
<i>ADC_CHANNEL_INTERNAL_TEMPSENSOR</i>	temperature sensor channel
<i>ADC_CHANNEL_INTERNAL_VREFINT</i>	vrefint channel
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value

<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC temperature sensor channel */
```

```
adc_internal_channel_config(ADC_CHANNEL_INTERNAL_TEMPSENSOR, ENABLE);
```

### adc\_sync\_mode\_config

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

**Table 3-12. Function adc\_sync\_mode\_config**

<b>Function name</b>	adc_sync_mode_config
<b>Function prototype</b>	void adc_sync_mode_config(uint32_t sync_mode);
<b>Function descriptions</b>	configure the ADC sync mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sync_mode</b>	ADC sync mode
<i>ADC_MODE_FREE</i>	free mode
<i>ADC_DAUL_ROUTINE _PARALLEL_INSERTE D_PARALLEL</i>	ADC0 and ADC1 work in combined routine parallel & inserted parallel mode
<i>ADC_DAUL_ROUTINE _PARALLEL_INSERTE D_ROTATION</i>	ADC0 and ADC1 work in combined routine parallel & trigger rotation mode
<i>ADC_DAUL_INSERTE D_PARALLEL_ROUTI NE_FOLLOWUP_FAST</i>	ADC0 and ADC1 work in combined inserted parallel + routine follow-up fast mode
<i>ADC_DAUL_INSERTE D_PARALLEL_ROUTI NE_FOLLOWUP_SLO W</i>	ADC0 and ADC1 work in combined inserted parallel + routine follow-up slow mode
<i>ADC_DAUL_INSERTE D_PARALLEL</i>	ADC0 and ADC1 work in inserted parallel mode
<i>ADC_DAUL_ROUTINE _PARALLEL</i>	ADC0 and ADC1 work in routine parallel mode
<i>ADC_DAUL_ROUTINE _FOLLOWUP_FAST</i>	ADC0 and ADC1 work in routine follow-up fast mode
<i>ADC_DAUL_ROUTINE</i>	ADC0 and ADC1 work in routine follow-up slow mode

<i>_FOLLOWUP_SLOW</i>	
<i>ADC_DAUL_INSERTE D_TRIGGER_ROTATI ON</i>	ADC0 and ADC1 work in inserted trigger rotation mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* ADC0 and ADC1 work in combined routine parallel & inserted parallel mode */
adc_sync_mode_config(ADC_DAUL_ROUTINE_PARALLEL_INSERTED_PARALLEL);
```

### adc\_sync\_routine\_data\_read

The description of `adc_sync_routine_data_read` is shown as below:

**Table 3-13. Function `adc_sync_routine_data_read`**

<b>Function name</b>	<code>adc_sync_routine_data_read</code>
<b>Function prototype</b>	<code>uint32_t adc_sync_routine_data_read(void);</code>
<b>Function descriptions</b>	read ADC0 and ADC1 routine data in sync mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<code>uint32_t</code>	the conversion value

Example:

```
/* read the last ADC0 and ADC1 conversion result data in sync mode */
uint32_t adc_value = 0;
adc_value = adc_sync_routine_data_read();
```

### adc\_data\_alignment\_config

The description of `adc_data_alignment_config` is shown as below:

**Table 3-14. 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 ADC data alignment

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0,1,2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>data_alignment</b>	data alignment select
<i>ADC_DATAALIGN_RIGHT</i>	right alignment
<i>ADC_DATAALIGN_LEFT</i>	left alignment
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-15. Function adc\_channel\_length\_config**

<b>Function name</b>	adc_channel_length_config
<b>Function prototype</b>	adc_channel_length_config(uint32_t adc_periph, uint8_t adc_sequence, uint32_t length);
<b>Function descriptions</b>	configure the length of routine sequence or inserted sequence
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0,1,2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
<i>ADC_ROUTINE_CHANNEL</i>	routine sequence
<i>ADC_INSERTED_CHANNEL</i>	inserted sequence
<b>Input parameter{in}</b>	
<b>length</b>	the length of the channel, routine sequence: 1-16, inserted channel 1-4
<b>Output parameter{out}</b>	



-	-
Return value	
-	-

Example:

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

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

### adc\_routine\_channel\_config

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

**Table 3-16. Function adc\_routine\_channel\_config**

<b>Function name</b>	adc_routine_channel_config
<b>Function prototype</b>	void adc_routine_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time)
<b>Function descriptions</b>	configure ADC routine channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
ADCx(x=0,1,2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>rank</b>	the routine sequence rank, this parameter must be between 0 to 15
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the selected ADC channel
ADC_CHANNEL_x	ADC_CHANNEL_x: x=0..17 for ADC0, x=0..17 for ADC1, x=0..16 for ADC2
<b>Input parameter{in}</b>	
<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	239.5 cycles

39POINT5	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 routine channel */
```

```
adc_routine_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 3-17. 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
ADCx(x=0,1,2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>rank</b>	the inserted sequencer rank, this parameter must be between 0 to 3
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the selected ADC channel
ADC_CHANNEL_x	ADC_CHANNEL_x: x=0..17 for ADC0, x=0..17 for ADC1, x=0..16 for ADC2
<b>Input parameter{in}</b>	
<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	71.5 cycles

1POINT5	
ADC_SAMPLETIME_2	239.5 cycles
39POINT5	
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

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

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

**Table 3-18. 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 sequence offset
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>adc_periph</b>	ADC peripheral
ADCx(x=0,1,2)	ADC peripheral selection
Input parameter{in}	
<b>inserted_channel</b>	inserted channel select
ADC_INSERTED_CHANNEL_x	inserted channel, x=0,1,2,3
Input parameter{in}	
<b>offset</b>	the offset data, this parameter must be between 0 to 4095
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### adc\_external\_trigger\_config

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

Table 3-19. Function `adc_external_trigger_config`

Function name	<code>adc_external_trigger_config</code>
Function prototype	<code>void adc_external_trigger_config(uint32_t adc_periph, uint8_t adc_sequence, uint32_t trigger_mode)</code>
Function descriptions	enable ADC external trigger
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC peripheral
<code>ADCx(x=0,1,2)</code>	ADC peripheral selection
Input parameter{in}	
<code>adc_sequence</code>	select the sequence
<code>ADC_ROUTINE_CHANNEL</code>	routine sequence
<code>ADC_INSERTED_CHANNEL</code>	inserted sequence
Input parameter{in}	
<code>trigger_mode</code>	select the trigger mode
<code>EXTERNAL_TRIGGER_DISABLE</code>	external trigger disable
<code>EXTERNAL_TRIGGER_RISING</code>	rising edge of external trigger
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC0 inserted sequence external trigger */
```

```
adc_external_trigger_config(ADC0, ADC_INSERTED_CHANNEL, EXTERNAL_TRIGGER_RISING);
```

### `adc_software_trigger_enable`

The description of `adc_software_trigger_enable` is shown as below:

Table 3-20. Function `adc_software_trigger_enable`

Function name	<code>adc_software_trigger_enable</code>
Function prototype	<code>void adc_software_trigger_enable(uint32_t adc_periph, uint8_t adc_sequence);</code>
Function descriptions	enable ADC software trigger
Precondition	-
The called functions	-
Input parameter{in}	

<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0,1,2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
<i>ADC_ROUTINE_CHAN NEL</i>	routine sequence
<i>ADC_INSERTED_CHA NNEL</i>	inserted sequence
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_software_trigger_enable(ADC0, ADC_ROUTINE_CHANNEL);
```

### adc\_routine\_data\_read

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

**Table 3-21. Function adc\_routine\_data\_read**

<b>Function name</b>	adc_routine_data_read
<b>Function prototype</b>	uint16_t adc_routine_data_read(uint32_t adc_periph);
<b>Function descriptions</b>	read ADC routine data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0,1,2)</i>	ADC peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	ADC conversion value (0 ~ 4095)

Example:

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

```
uint16_t adc_value = 0;
```

```
adc_value = adc_routine_data_read(ADC0);
```

### adc\_inserted\_data\_read

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

Table 3-22. Function `adc_inserted_data_read`

Function name	<code>adc_inserted_data_read</code>
Function prototype	<code>uint16_t adc_inserted_data_read(uint32_t adc_periph);</code>
Function descriptions	read ADC inserted data register
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC peripheral
<code>ADCx(x=0,1,2)</code>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
<code>uint16_t</code>	ADC conversion value (0 ~ 4095)

Example:

```
/* read ADC0 inserted sequence register */
uint16_t adc_value = 0;
adc_value = adc_inserted_data_read (ADC0);
```

### `adc_latch_data_read`

The description of `adc_latch_data_read` is shown as below:

Table 3-23. Function `adc_latch_data_read`

Function name	<code>adc_latch_data_read</code>
Function prototype	<code>uint16_t adc_latch_data_read(uint32_t adc_periph, uint8_t latch_data);</code>
Function descriptions	read ADC latch data register
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC peripheral
<code>ADCx(x=0,1,2)</code>	ADC peripheral selection
Input parameter{in}	
<code>latch_data</code>	Latch data selection
<code>ADC_LATCH_DATA_x</code>	latch data x, x=0,1,2,3
Output parameter{out}	
-	-
Return value	
<code>uint16_t</code>	ADC conversion value (0 ~ 4095)

Example:

```
/* read ADC0 latch data register */
uint16_t adc_value = 0;
```

```
adc_value = adc_latch_data_read(ADC0, ADC_LATCH_DATA_0);
```

### adc\_latch\_data\_source\_config

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

**Table 3-24. Function adc\_latch\_data\_source\_config**

<b>Function name</b>	adc_latch_data_source_config
<b>Function prototype</b>	void adc_latch_data_source_config(uint32_t adc_periph, uint8_t latch_data, uint8_t adc_sequence, uint8_t rank);
<b>Function descriptions</b>	configure ADC latch data source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
ADCx(x=0,1,2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>latch_data</b>	Latch data selection
ADC_LATCH_DATA_x	latch data x, x=0,1,2,3
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	select the sequence
ADC_ROUTINE_CHAN NEL	routine sequence
ADC_INSERTED_CHA NNEL	inserted sequence
<b>Input parameter{in}</b>	
<b>rank</b>	the routine sequence rank, this parameter must be between 0 to 15 the inserted sequence rank, this parameter must be between 0 to 3
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_latch_data_source_config(ADC0, ADC_LATCH_DATA_0, ADC_ROUTINE_CHANNEL,  
5);
```

### adc\_watchdog0\_single\_channel\_enable

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

**Table 3-25. Function adc\_watchdog0\_single\_channel\_enable**

<b>Function name</b>	adc_watchdog0_single_channel_enable
<b>Function prototype</b>	void adc_watchdog0_single_channel_enable(uint32_t adc_periph, uint8_t

	adc_channel);
<b>Function descriptions</b>	configure ADC analog watchdog0 single channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0,1,2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	the selected ADC channel
<i>ADC_CHANNEL_x</i>	ADC_CHANNEL_x: x=0..17 for ADC0, x=0..17 for ADC1, x=0..16 for ADC2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC0 analog watchdog0 single channel */
```

```
adc_watchdog0_single_channel_enable(ADC0, ADC_CHANNEL_1);
```

### adc\_watchdog0\_sequence\_channel\_enable

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

**Table 3-26. Function adc\_watchdog0\_sequence\_channel\_enable**

<b>Function name</b>	adc_watchdog0_sequence_channel_enable
<b>Function prototype</b>	void adc_watchdog0_sequence_channel_enable(uint32_t adc_periph, uint8_t adc_sequence);
<b>Function descriptions</b>	configure ADC analog watchdog0 sequence channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0,1,2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_sequence</b>	the sequence use analog watchdog
<i>ADC_ROUTINE_CHANNEL</i>	routine sequence
<i>ADC_INSERTED_CHANNEL</i>	inserted sequence
<i>ADC_ROUTINE_INSERTED_CHANNEL</i>	both routine and inserted sequence
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	



-	-
---	---

Example:

```
/* configure ADC analog watchdog0 sequence */
```

```
adc_watchdog0_sequence_channel_enable(ADC0, ADC_ROUTINE_CHANNEL);
```

### adc\_watchdog0\_disable

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

**Table 3-27. Function adc\_watchdog0\_disable**

<b>Function name</b>	adc_watchdog0_disable
<b>Function prototype</b>	void adc_watchdog0_disable(uint32_t adc_periph);
<b>Function descriptions</b>	disable ADC analog watchdog 0
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<b>ADCx(x=0,1,2)</b>	ADC peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC0 analog watchdog0 */
```

```
adc_watchdog0_disable(ADC0);
```

### adc\_watchdog0\_threshold\_config

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

**Table 3-28. Function adc\_watchdog0\_threshold\_config**

<b>Function name</b>	adc_watchdog0_threshold_config
<b>Function prototype</b>	void adc_watchdog0_threshold_config(uint32_t adc_periph, uint32_t low_threshold, uint32_t high_threshold);
<b>Function descriptions</b>	configure ADC analog watchdog 0 threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<b>ADCx(x=0,1,2)</b>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>low_threshold</b>	analog watchdog low threshold, 0..2 <sup>20</sup> -1

Input parameter{in}	
<b>high_threshold</b>	analog watchdog high threshold, 0..2 <sup>20</sup> -1
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_watchdog0_threshold_config(ADC0, 0x0400, 0x0A00);
```

### adc\_resolution\_config

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

**Table 3-29. Function adc\_resolution\_config**

<b>Function name</b>	adc_resolution_config
<b>Function prototype</b>	void adc_resolution_config(uint32_t adc_periph, uint32_t resolution);
<b>Function descriptions</b>	configure ADC resolution
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>adc_periph</b>	ADC peripheral
ADCx(x=0,1,2)	ADC peripheral selection
Input parameter{in}	
<b>resolution</b>	ADC resolution
ADC_RESOLUTION_12B	12-bit ADC resolution
ADC_RESOLUTION_10B	10-bit ADC resolution
ADC_RESOLUTION_8B	8-bit ADC resolution
ADC_RESOLUTION_6B	6-bit ADC resolution
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 resolution */
```

```
adc_resolution_config(ADC0, ADC_RESOLUTION_12B);
```

## adc\_oversample\_mode\_config

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

**Table 3-30. 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=0,1,2)	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_SHIFT_NONE	no oversampling shift
ADC_OVERSAMPLING_SHIFT_1B	1-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_2B	2-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_3B	3-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_4B	4-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_5B	5-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_6B	6-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_7B	7-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_8B	8-bit oversampling shift
<b>Input parameter{in}</b>	
<b>ratio</b>	ADC oversampling ratio
ADC_OVERSAMPLING_RATIO_MUL2	oversampling ratio multiple 2

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

Example:

```
/* configure ADC0 oversample mode: 16 times sample, 4 bits shift */
```

```
adc_oversample_mode_config( ADC0, 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 3-31. Function adc\_oversample\_mode\_enable**

<b>Function name</b>	adc_oversample_mode_enable
<b>Function prototype</b>	void adc_oversample_mode_enable(uint32_t adc_periph);
<b>Function descriptions</b>	enable ADC oversample mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>adc_periph</b>	ADC peripheral
ADCx(x=0,1,2)	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 3-32. 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=0,1,2)</i>	ADC peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC0 oversample mode */
```

```
adc_oversample_mode_disable (ADC0);
```

### adc\_flag\_get

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

**Table 3-33. Function adc\_flag\_get**

<b>Function name</b>	adc_flag_get
<b>Function prototype</b>	FlagStatus adc_flag_get(uint32_t adc_periph, uint32_t flag);
<b>Function descriptions</b>	get the ADC flag bits
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0,1,2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>flag</b>	the adc flag bits
<i>ADC_FLAG_WD0E</i>	analog watchdog0 event flag
<i>ADC_FLAG_EORC</i>	end of routine sequence conversion flag
<i>ADC_FLAG_EOIC</i>	end of inserted sequence conversion flag
<i>ADC_FLAG_STIC</i>	start of inserted sequence conversion flag
<i>ADC_FLAG_STRC</i>	start of routine sequence conversion flag
<b>Output parameter{out}</b>	

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

Example:

```
/* get the ADC0 analog watchdog0 flag bits*/
```

```
FlagStatus flag_value;
```

```
flag_value = adc_flag_get(ADC0, ADC_FLAG_WD0E);
```

### adc\_flag\_clear

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

**Table 3-34. Function adc\_flag\_clear**

<b>Function name</b>	adc_flag_clear
<b>Function prototype</b>	void adc_flag_clear(uint32_t adc_periph, uint32_t flag);
<b>Function descriptions</b>	clear the ADC flag bits
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
ADCx(x=0,1,2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>flag</b>	the adc flag bits
ADC_FLAG_WD0E	analog watchdog0 event flag
ADC_FLAG_EORC	end of routine sequence conversion flag
ADC_FLAG_EOIC	end of inserted sequence conversion flag
ADC_FLAG_STIC	start of inserted sequence conversion flag
ADC_FLAG_STRC	start of routine sequence conversion flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the ADC0 analog watchdog0 flag bits*/
```

```
adc_flag_clear(ADC0, ADC_FLAG_WD0E);
```

### adc\_interrupt\_enable

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

**Table 3-35. 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 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=0,1,2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	the adc interrupt
<i>ADC_INT_WD0E</i>	analog watchdog 0 interrupt
<i>ADC_INT_EORC</i>	end of routine sequence conversion interrupt
<i>ADC_INT_EOIC</i>	end of inserted sequence conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC0 analog watchdog0 interrupt */
```

```
adc_interrupt_enable(ADC0, ADC_INT_WD0E);
```

### adc\_interrupt\_disable

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

**Table 3-36. 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 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=0,1,2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	the adc interrupt
<i>ADC_INT_WD0E</i>	analog watchdog 0 interrupt
<i>ADC_INT_EORC</i>	end of routine sequence conversion interrupt
<i>ADC_INT_EOIC</i>	end of inserted sequence conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC0 interrupt */
adc_interrupt_disable(ADC0, ADC_INT_WD0E);
```

### adc\_interrupt\_flag\_get

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

**Table 3-37. 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 flag);
<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
ADCx(x=0,1,2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>flag</b>	the adc interrupt bits
ADC_INT_FLAG_WD0E	analog watchdog 0 interrupt flag
ADC_INT_FLAG_EORC	end of routine sequence conversion interrupt flag
ADC_INT_FLAG_EOIC	end of inserted sequence conversion interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the ADC analog watchdog 0 interrupt bits*/
FlagStatus flag_value;
flag_value = adc_interrupt_flag_get(ADC0, ADC_INT_FLAG_WD0E);
```

### adc\_interrupt\_flag\_clear

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

**Table 3-38. 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 flag);
<b>Function descriptions</b>	clear the ADC interrupt bits
<b>Precondition</b>	-



The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0,1,2)	ADC peripheral selection
Input parameter{in}	
flag	the adc interrupt bits
ADC_INT_FLAG_WD0E	analog watchdog 0 interrupt flag
ADC_INT_FLAG_EORC	end of routine sequence conversion interrupt flag
ADC_INT_FLAG_EOIC	end of inserted sequence conversion interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the ADC0 analog watchdog0 interrupt bits*/
```

```
adc_interrupt_flag_clear(ADC0, ADC_INT_FLAG_WD0E);
```

### 3.3. BKP

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

#### 3.3.1. Descriptions of Peripheral registers

BKP registers are listed in the table shown as below:

**Table 3-39. 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 3-40. BKP firmware function**

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

#### bkp\_deinit

The description of bkp\_deinit is shown as below:

**Table 3-41. Function bkp\_deinit**

Function name	bkp_deinit
---------------	------------

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

Example:

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

```
bkp_deinit ();
```

### bkp\_write\_data

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

**Table 3-42. Function bkp\_write\_data**

<b>Function name</b>	bkp_write_data
<b>Function prototype</b>	void bkp_write_data(bkp_data_register_enum register_number, uint16_t data);
<b>Function descriptions</b>	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
0-0xffff	data value

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write BKP data register */
bkp_write_data (BKP_DATA_0, 0x1226);
```

### bkp\_read\_data

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

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

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

Example:

```
/* read BKP data register */
uint16_t data;
data = bkp_read_data (BKP_DATA_0);
```

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

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

**Table 3-44. Function bkp\_rtc\_calibration\_output\_enable**

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

```
/* 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 3-45. Function bkp\_rtc\_calibration\_output\_disable**

<b>Function name</b>	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 3-46. Function bkp\_rtc\_signal\_output\_enable**

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

Example:

```
/* enable RTC alarm or second signal output */
bkp_rtc_signal_output_enable();
```

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

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

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

Function name	bkp_rtc_signal_output_disable
Function prototype	void bkp_rtc_signal_output_disable (void);
Function descriptions	disable RTC alarm or second signal output
Precondition	-
The called functions	-

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

Example:

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

```
bkp_rtc_signal_output_disable();
```

### bkp\_rtc\_output\_select

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

**Table 3-48. Function bkp\_rtc\_output\_select**

Function name	bkp_rtc_output_select
Function prototype	void bkp_rtc_output_select (uint16_t outputsel);
Function descriptions	select RTC output, the RTC output can be select as alarm pulse or second pulse
Precondition	-
The called functions	-
Input parameter{in}	
outputsel	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 3-49. 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>	-
<b>Input parameter{in}</b>	
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-50. 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
<i>RTC_CLOCK_SLOWED_DOWN</i>	RTC clock slowed down
<i>RTC_CLOCK_SPEED_UP</i>	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 3-51. Function bkp\_rtc\_calibration\_value\_set**

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

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 3-52. 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 3-53. 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 3-54. Function bkp\_tamper\_active\_level\_set**

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

Example:

```
/* set 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 3-55. 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 3-56. Function bkp\_tamper\_interrupt\_disable**

<b>Function name</b>	bkp_tamper_interrupt_disable
<b>Function prototype</b>	void bkp_tamper_interrupt_disable (void);
<b>Function descriptions</b>	disable tamper interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-

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 3-57. Function bkp\_flag\_get**

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

Example:

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

### bkp\_flag\_clear

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

Table 3-58. Function bkp\_flag\_clear

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

Example:

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

```
bkp_flag_clear (BKP_FLAG_TAMPER);
```

### bkp\_interrupt\_flag\_get

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

Table 3-59. Function bkp\_interrupt\_flag\_get

Function name	bkp_interrupt_flag_get
Function prototype	FlagStatus bkp_interrupt_flag_get(uint16_t flag);
Function descriptions	get bkp interrupt flag state
Precondition	-
The called functions	-
Input parameter{in}	
flag	bkp interrupt flag state
BKP_INT_FLAG_TAMPER	tamper interrupt flag

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

Example:

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

```
bkp_interrupt_flag_get (BKP_INT_FLAG_TAMPER);
```

### bkp\_interrupt\_flag\_clear

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

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

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

Example:

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

```
bkp_interrupt_flag_clear (BKP_INT_FLAG_TAMPER);
```

## 3.4. CAN

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

### 3.4.1. Descriptions of Peripheral registers

CAN registers are listed in the table shown as below:

**Table 3-61. CAN Registers**

Registers	Descriptions
CAN_CTL	Control register
CAN_STAT	Status register
CAN_TSTAT	Transmit status register
CAN_RFIFO0	Receive message FIFO0 register
CAN_RFIFO1	Receive message FIFO1 register
CAN_INTEN	Interrupt enable register
CAN_ERR	Error register
CAN_BT	Bit timing register
CAN_FDCTL	FD control register
CAN_FDSTAT	FD status register
CAN_FDTDC	FD transmitter delay compensation register
CAN_DBT	Date Bit timing register
CAN_TMIx	Transmit mailbox identifier register
CAN_TMPx	Transmit mailbox property register
CAN_TMDATA0x	Transmit mailbox data0 register
CAN_TMDATA1x	Transmit mailbox data1 register
CAN_RFIFOMIx	Receive FIFO mailbox identifier register
CAN_RFIFOMPx	Receive FIFO mailbox property register
CAN_RFIFOMDAT A0x	Receive FIFO mailbox data0 register
CAN_RFIFOMDAT A1x	Receive FIFO mailbox data1 register
CAN_FCTL	Filter control register
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 3-62. 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_fd_init	initialize CAN FD function
can_filter_init	initialize CAN filter
can_filter_mask_mode_init	CAN filter mask mode initialization
can_monitor_mode_set	CAN communication mode configure
can_fd_function_enable	CAN FD frame function enable
can_fd_function_disable	CAN FD frame function disable
can1_filter_start_bank	set can1 filter start bank number
can_debug_freeze_enable	CAN debug freeze enable
can_debug_freeze_disable	CAN debug freeze disable
can_time_trigger_mode_enable	CAN time trigger mode enable
can_time_trigger_mode_disable	CAN time trigger mode disable
can_message_transmit	transmit CAN message
can_transmit_states	get CAN transmit state
can_transmission_stop	stop CAN transmission
can_message_receive	CAN receive message
can_fifo_release	CAN release fifo
can_receive_message_length_get	CAN receive message length
can_working_mode_set	CAN working mode
can_wakeup	CAN wakeup from sleep mode
can_error_get	CAN get error
can_receive_error_number_get	get CAN receive error number
can_transmit_error_number_get	get CAN transmit error number
can_interrupt_enable	CAN interrupt enable
can_interrupt_disable	CAN interrupt disable
can_flag_get	CAN get flag state
can_flag_clear	CAN clear flag state
can_interrupt_flag_get	CAN get interrupt flag state
can_interrupt_flag_clear	CAN clear interrupt flag state
can_fifo_overrun_flag_get	get FIFO overrun flag state

#### Structure can\_parameter\_struct

**Table 3-63. Structure can\_parameter\_struct**

Member name	Function description
working_mode	CAN working mode

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

### Structure can\_transmit\_message\_struct

**Table 3-64. Structure can\_transmit\_message\_struct**

Member name	Function description
tx_sfid	standard format frame identifier
tx_efid	extended format frame identifier
tx_ff	format of frame, standard or extended format
tx_ft	type of frame, data or remote
tx_dlen	data length
tx_data[64]	transmit data
fd_flag	CAN FD frame flag
fd_brs	bit rate of data switch
fd_esi	error status indicator

### Structure can\_receive\_message\_struct

**Table 3-65. 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[64]	receive data
rx_fi	filtering index
fd_flag	CAN FD frame flag
fd_brs	bit rate of data switch
fd_esi	error status indicator

## Structure can\_filter\_parameter\_struct

**Table 3-66. 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

## Structure can\_fd\_tdc\_struct

**Table 3-67. Structure can\_fd\_tdc\_struct**

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

## Structure can\_fdframe\_struct

**Table 3-68. Structure can\_fdframe\_struct**

Member name	Function description
fd_frame	FD operation function
excp_event_detect	protocol exception event detection function
delay_compensation	transmitter delay compensation mode
p_delay_compensation	pointer to the struct of the transmitter delay compensation, refer to <a href="#">Table 3-67. Structure can_fd_tdc_struct</a>
edge_filter_disable	edge filtering disable
iso_bosch	ISO/Bosch mode choice
esi_mode	error state indicator mode
data_resync_jump_width	CAN resynchronization jump width
data_time_segment_1	time segment 1
data_time_segment_2	time segment 2
data_prescaler	baudrate prescaler

## can\_deinit

The description of can\_deinit is shown as below:

Table 3-69. Function can\_deinit

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

Example:

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

### can\_struct\_para\_init

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

Table 3-70. Function can\_struct\_para\_init

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

Example:

```
/* Initialize CAN parameter struct */
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 3-71. 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
<b>CANx(x=0,1)</b>	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 3-63. Structure can_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS / ERROR

Example:

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

## can\_fd\_init

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

**Table 3-72. Function can\_fd\_init**

<b>Function name</b>	can_fd_init
<b>Function prototype</b>	ErrStatus can_fd_init(uint32_t can_periph, can_fdframe_struct* can_fdframe_init);
<b>Function descriptions</b>	initialize CAN FD function
<b>Precondition</b>	can_struct_para_init()

The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
can_fdframe_init	parameters for CAN FD initialization, the structure members can refer to members of the structure <a href="#">Table 3-68. Structure can_fdframe_struct</a>
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR

Example:

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

### can\_filter\_init

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

**Table 3-73. Function can\_filter\_init**

Function name	can_filter_init
Function prototype	void can_filter_init(can_filter_parameter_struct* can_filter_parameter_init);
Function descriptions	initialize CAN filter
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_filter_parameter_init	CAN filter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-66. Structure can_filter_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize CAN filter */
can_filter_init(&can_filter);
```

### can\_filter\_mask\_mode\_init

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

Table 3-74. Function `can_filter_mask_mode_init`

Function name	<code>can_filter_mask_mode_init</code>
Function prototype	<code>void can_filter_mask_mode_init(uint32_t id, uint32_t mask, can_format_fifo_enum format_fifo, uint16_t filter_number)</code>
Function descriptions	CAN filter mask mode initialization
Precondition	-
The called functions	<code>can_filter_init()</code>
Input parameter{in}	
id	value range (0x00000000 - 0xFFFFFFFF)
Input parameter{in}	
mask	value range (0x00000000 - 0xFFFFFFFF)
Input parameter{in}	
format_fifo	format and fifo states, only one parameter can be selected which is shown as below
<code>CAN_STANDARD_FIFO0</code>	standard format and store to FIFO0
<code>CAN_STANDARD_FIFO1</code>	standard format and store to FIFO1
<code>CAN_EXTENDED_FIFO0</code>	extended format and store to FIFO0
<code>CAN_EXTENDED_FIFO1</code>	extended format and store to FIFO1
Input parameter{in}	
filter_number	filter sequence number, value range(0x00 - 0x1B)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CAN filter mask mode initialization */
can_filter_mask_mode_init(0x11, 0x11, CAN_STANDARD_FIFO0, 0);
```

### `can_monitor_mode_set`

The description of `can_monitor_mode_set` is shown as below:

Table 3-75. Function `can_monitor_mode_set`

Function name	<code>can_monitor_mode_set</code>
Function prototype	<code>ErrStatus can_monitor_mode_set(uint32_t can_periph, uint8_t mode)</code>
Function descriptions	CAN communication mode configure
Precondition	-
The called functions	-
Input parameter{in}	

<b>can_periph</b>	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>mode</b>	communication mode, only one parameter can be selected which is shown as below
<i>CAN_NORMAL_MODE</i>	normal mode
<i>CAN_LOOPBACK_MODE</i>	loopback mode
<i>CAN_SILENT_MODE</i>	silent mode
<i>CAN_SILENT_LOOPBACK_MODE</i>	silent loopback mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS / ERROR

Example:

```
/* CAN communication mode configure */
can_monitor_mode_set(CAN0, CAN_NORMAL_MODE);
```

### can\_fd\_function\_enable

The description of can\_fd\_function\_enable is shown as below:

**Table 3-76. Function can\_fd\_function\_enable**

<b>Function name</b>	can_fd_function_enable
<b>Function prototype</b>	void can_fd_function_enable(uint32_t can_periph)
<b>Function descriptions</b>	CAN FD frame function enable
<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:

```
/* CAN0 FD frame function enable */
can_fd_function_enable(CAN0);
```



## can\_fd\_function\_disable

The description of can\_fd\_function\_disable is shown as below:

**Table 3-77. Function can\_fd\_function\_disable**

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

Example:

```
/* CAN0 FD frame function disable */
can_fd_function_disable(CAN0);
```

## can1\_filter\_start\_bank

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

**Table 3-78. 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 fliter start bank number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>start_bank</b>	CAN1 start bank number
<b>1..27</b>	start number
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-79. 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
<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 3-80. 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>	
-	-

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 3-81. 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>	
-	-

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 3-82. 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
<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 3-83. 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 3-64. Structure can_transmit_message_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 3-84. 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
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>mailbox_number</b>	Mailbox number

<i>CAN_MAILBOXx</i>	CAN_MAILBOXx(x=0,1,2)
<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 3-85. Function can\_transmission\_stop**

<b>Function name</b>	can_transmission_stop
<b>Function prototype</b>	ErrStatus 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>	CAN_MAILBOXx(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS / ERROR

Example:

```
/* stop CAN0 mailbox0 transmission */
```

```
can_transmission_stop (CAN0, CAN_MAILBOX0);
```

### can\_message\_receive

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

**Table 3-86. 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 3-65. Structure can_receive_message_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-87. Function can\_fifo\_release**

<b>Function name</b>	can_fifo_release
<b>Function prototype</b>	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 3-88. 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
<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>	
<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 3-89. 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
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>can_working_mode</b>	Mode select
<i>CAN_MODE_INITIALIZE</i>	Initialize mode
<i>CAN_MODE_NORMAL</i>	Normal mode
<i>CAN_MODE_SLEEP</i>	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 3-90. 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

Example:

```
/* wake up CAN0 */
```

```
can_wakeup (CAN0);
```

### can\_error\_get

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



Table 3-91. 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

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 3-92. Function can\_receive\_error\_number\_get

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

Example:

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

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

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

Table 3-93. Function `can_transmit_error_number_get`

<b>Function name</b>	<code>can_transmit_error_number_get</code>
<b>Function prototype</b>	<code>uint8_t can_transmit_error_number_get(uint32_t can_periph);</code>
<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

Example:

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

## can\_flag\_get

The description of `can_flag_get` is shown as below:

Table 3-94. Function `can_flag_get`

<b>Function name</b>	<code>can_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus can_flag_get(uint32_t can_periph, can_flag_enum flag);</code>
<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>	
<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
<i>CAN_FLAG_BOERR</i>	bus-off error

<code>CAN_FLAG_PERR</code>	passive error
<code>CAN_FLAG_WERR</code>	warning error
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-95. Function can\_flag\_clear**

<b>Function name</b>	can_flag_clear
<b>Function prototype</b>	void can_flag_clear(uint32_t can_periph, can_flag_enum flag);
<b>Function descriptions</b>	clear CAN flag state
<b>Precondition</b>	-
<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>flag</b>	CAN flags
<code>CAN_FLAG_MTE2</code>	mailbox 2 transmit error
<code>CAN_FLAG_MTE1</code>	mailbox 1 transmit error
<code>CAN_FLAG_MTE0</code>	mailbox 0 transmit error
<code>CAN_FLAG_MTF2</code>	mailbox 2 transmit finished
<code>CAN_FLAG_MTF1</code>	mailbox 1 transmit finished
<code>CAN_FLAG_MTF0</code>	mailbox 0 transmit finished
<code>CAN_FLAG_RFO0</code>	receive FIFO0 overfull
<code>CAN_FLAG_RFF0</code>	receive FIFO0 full
<code>CAN_FLAG_RFO1</code>	receive FIFO1 overfull
<code>CAN_FLAG_RFF1</code>	receive FIFO1 full
<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\_enable

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

**Table 3-96. 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
CAN_INT_ERR	error interrupt enable
CAN_INT_WU	wakeup interrupt enable
CAN_INT_SLPW	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 3-97. Function `can_interrupt_disable`

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

Example:

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

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

### `can_interrupt_flag_get`

The description of `can_interrupt_flag_get` is shown as below:

Table 3-98. Function `can_interrupt_flag_get`

Function name	<code>can_interrupt_flag_get</code>
Function prototype	<code>FlagStatus can_interrupt_flag_get(uint32_t can_periph, can_interrupt_flag_enum flag);</code>
Function descriptions	get CAN interrupt 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 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 3-99. 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
<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>F</i>	
<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>	
-	-

Example:

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

### can\_fifo\_overnrun\_flag\_get

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

**Table 3-100. Function can\_fifo\_overnrun\_flag\_get**

<b>Function name</b>	can_fifo_overnrun_flag_get
<b>Function prototype</b>	FlagStatus can_fifo_overnrun_flag_get(uint32_t can_periph, uint8_t fifo_number);
<b>Function descriptions</b>	get FIFO overrun 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>fifo_number</b>	Fifo number
<i>CAN_FIFOx</i>	CAN_FIFOx(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get FIFO0 overrun flag state of CAN0 */
```

```
can_fifo_overnun_flag_get(CAN0, CAN_FIFO0);
```

## 3.5. CAU

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

### 3.5.1. Descriptions of Peripheral registers

CAU registers are listed in the table shown as below:

**Table 3-101. CAU Registers**

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

### 3.5.2. Descriptions of Peripheral functions

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

**Table 3-102. CAU firmware function**

Function name	Function description
cau_deinit	reset the CAU peripheral
cau_struct_para_init	initialize the CAU encrypt and decrypt parameter struct with the default values
cau_key_struct_para_init	initialize the key parameter struct with the default values
cau_enable	enable the CAU peripheral
cau_disable	disable the CAU peripheral
cau_dma_enable	enable the CAU DMA interface



Function name	Function description
cau_dma_disable	disable the CAU DMA interface
cau_init	initialize the CAU peripheral
cau_aes_keysize_config	configure key size if use AES algorithm
cau_key_init	initialize the key parameters
cau_fifo_flush	flush the IN and OUT FIFOs
cau_enable_state_get	return whether CAU peripheral is enabled or disabled
cau_data_write	write data to the IN FIFO
cau_data_read	return the last data entered into the output FIFO
cau_aes_ecb	encrypt and decrypt using AES in ECB mode
cau_flag_get	get the CAU flag status
cau_interrupt_enable	enable the CAU interrupts
cau_interrupt_disable	disable the CAU interrupts
cau_interrupt_flag_get	get the interrupt flag

### Structure cau\_key\_parameter\_struct

**Table 3-103. Structure cau\_key\_parameter\_struct**

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

### Structure cau\_parameter\_struct

**Table 3-104. Structure cau\_parameter\_struct**

Member name	Function description
alg_dir	algorithm directory
*key	key
key_size	key size in bytes
*input	input data
in_length	input data length in bytes

### cau\_deinit

The description of cau\_deinit is shown as below:

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

Function name	cau_deinit
Function prototype	void cau_deinit(void)

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

Example:

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

```
cau_deinit();
```

**cau\_struct\_para\_init**

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

**Table 3-106. Function cau\_struct\_para\_init**

<b>Function name</b>	cau_struct_para_init
<b>Function prototype</b>	void cau_struct_para_init(cau_parameter_struct *cau_parameter)
<b>Function descriptions</b>	initialize the CAU encrypt and decrypt parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>cau_parameter</b>	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-104. Structure cau_parameter_struct</a>
<b>Return value</b>	
-	-

Example:

```
cau_parameter_struct text;
```

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

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

**cau\_key\_struct\_para\_init**

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

**Table 3-107. Function cau\_key\_struct\_para\_init**

<b>Function name</b>	cau_key_struct_para_init
----------------------	--------------------------

<b>Function prototype</b>	void cau_key_struct_para_init(cau_key_parameter_struct *key_initpara)
<b>Function descriptions</b>	initialize the key parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
key_initpara	the key parameters, refer to structure <a href="#">Table 3-103. Structure cau_key_parameter_struct</a>
<b>Return value</b>	
-	-

Example:

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

## cau\_enable

The description of cau\_enable is shown as below:

**Table 3-108. Function cau\_enable**

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

Example:

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

## cau\_disable

The description of cau\_disable is shown as below:

**Table 3-109. Function cau\_disable**

<b>Function name</b>	cau_disable
----------------------	-------------

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

Example:

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

```
cau_disable();
```

### cau\_dma\_enable

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

**Table 3-110. Function cau\_dma\_enable**

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

Example:

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

```
cau_dma_enable(CAU_DMA_INFIFO);
```

### cau\_dma\_disable

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

**Table 3-111. Function cau\_dma\_disable**

<b>Function name</b>	cau_dma_disable
----------------------	-----------------

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

Example:

```
/* disable the CAU DMA interface */
cau_dma_disable(CAU_DMA_INFIFO);
```

## cau\_init

The description of cau\_init is shown as below:

**Table 3-112. Function cau\_init**

<b>Function name</b>	cau_init
<b>Function prototype</b>	void cau_init(uint32_t alg_dir, uint32_t algo_mode, uint32_t swapping)
<b>Function descriptions</b>	initialize the CAU peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>alg_dir</b>	algorithm direction
<i>CAU_ENCRYPT</i>	encrypt
<i>CAU_DECRYPT</i>	decrypt
<b>Input parameter{in}</b>	
<b>algo_mode</b>	algorithm mode selection
<i>CAU_MODE_AES_EC B</i>	AES-ECB (AES Electronic codebook)
<i>CAU_MODE_AES_KE Y</i>	AES decryption key preparation mode
<b>Input parameter{in}</b>	
<b>swapping</b>	data swapping selection
<i>CAU_SWAPPING_32BIT</i>	no swapping
<i>CAU_SWAPPING_16BIT</i>	half-word swapping
<i>CAU_SWAPPING_8BIT</i>	bytes swapping

CAU_SWAPPING_1BIT	bit swapping
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### cau\_aes\_keysize\_config

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

**Table 3-113. Function cau\_aes\_keysize\_config**

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

Example:

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

```
cau_aes_keysize_config(CAU_KEYSIZE_128BIT);
```

### cau\_key\_init

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

**Table 3-114. Function cau\_key\_init**

Function name	cau_key_init
Function prototype	void cau_key_init(cau_key_parameter_struct* key_initpara);
Function descriptions	initialize the key parameters
Precondition	-
The called functions	-

Input parameter{in}	
key_initpara	the key parameters, refer to structure <a href="#">Table 3-103. Structure cau_key_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the key parameters */
cau_key_parameter_struct key_initpara;
cau_key_init (&key_initpara);
```

### cau\_fifo\_flush

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

**Table 3-115. Function cau\_fifo\_flush**

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

Example:

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

### cau\_enable\_state\_get

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

**Table 3-116. Function cau\_enable\_state\_get**

Function name	cau_enable_state_get
Function prototype	ControlStatus cau_enable_state_get(void);
Function descriptions	return whether CAU peripheral is enabled or disabled
Precondition	-
The called functions	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ControlStatus	ENABLE or DISABLE

Example:

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

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

### cau\_data\_write

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

**Table 3-117. Function cau\_data\_write**

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

Example:

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

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

### cau\_data\_read

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

**Table 3-118. Function cau\_data\_read**

Function name	cau_data_read
Function prototype	uint32_t cau_data_read(void);
Function descriptions	return the last data entered into the output FIFO
Precondition	-
The called functions	-
Input parameter{in}	
-	-



Output parameter{out}	
-	-
Return value	
uint32_t	0x0 – 0xFFFFFFFF

Example:

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

```
uint32_t data;
```

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

### cau\_aes\_ecb

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

**Table 3-119. Function cau\_aes\_ecb**

Function name	cau_aes_ecb
Function prototype	ErrStatus cau_aes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);
Function descriptions	encrypt and decrypt using AES in ECB mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	the CAU encrypt and decrypt parameters, refer to structure <a href="#">Table 3-104. Structure cau_parameter_struct</a>
Output parameter{out}	
output	pointer to the returned buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
cau_parameter_struct text;
```

```
uint8_t encrypt_result[TEXT_SIZE];
```

```
ErrStatus status;
```

```
.....
```

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

```
key_addr = key_select[i];
```

```
key_size = keysize[i];
```

```
text.alg_dir = CAU_ENCRYPT;
```

```
text.key = key_addr;
```

```

text.key_size = key_size;

text.input     = plaintext;

text.in_length = TEXT_SIZE;

/* encryption in ECB mode */

status = cau_aes_ecb(&text, encrypt_result);

```

### cau\_flag\_get

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

**Table 3-120. Function cau\_flag\_get**

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

Example:

```

/* get the CAU flag status */

FlagStatus status;

status = cau_flag_get (CAU_FLAG_INFIFO_EMPTY);

```

### cau\_interrupt\_enable

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

**Table 3-121. Function cau\_interrupt\_enable**

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

Example:

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

### cau\_interrupt\_disable

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

**Table 3-122. Function cau\_interrupt\_disable**

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

Example:

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

## cau\_interrupt\_flag\_get

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

**Table 3-123. Function cau\_interrupt\_flag\_get**

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

Example:

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

```
FlagStatus status = cau_interrupt_flag_get(CAU_INT_FLAG_INFIFO);
```

## 3.6. CMP

The general purpose comparator can work either standalone(all terminal are available on I/Os) or together with the timers. It provide a trigger source when an analog signal is in a certain condition, achieves some current control by working together with a PWM output of a TIMER. The CMP registers are listed in chapter [3.6.1](#), the CMP firmware functions are introduced in chapter [3.6.2](#).

### 3.6.1. Descriptions of Peripheral registers

CMP registers are listed in the table shown as below:

**Table 3-124. CMP registers**

Registers	Descriptions
CMP_STAT	CMP status register
CMP_IFC	CMP interrupt flag clear register
CMP0_CS	CMP0 control and status register

### 3.6.2. Descriptions of Peripheral functions

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

**Table 3-125. CMP firmware function**

Function name	Function description
cmp_deinit	CMP deinit
cmp_mode_init	CMP mode init
cmp_output_init	CMP output init
cmp_blanking_init	CMP output blanking function init
cmp_digital_filter_init	CMP digital filter init
cmp_enable	enable CMP
cmp_disable	disable CMP
cmp_lock_enable	lock the CMP
cmp_voltage_scaler_enable	enable the voltage scaler
cmp_voltage_scaler_disable	disable the voltage scaler
cmp_scaler_bridge_enable	enable the scaler bridge
cmp_scaler_bridge_disable	disable the scaler bridge
cmp_output_level_get	get output level
cmp_flag_get	get CMP flag
cmp_flag_clear	clear CMP flag
cmp_interrupt_enable	enable CMP interrupt
cmp_interrupt_disable	disable CMP interrupt
cmp_interrupt_flag_get	get CMP interrupt flag
cmp_interrupt_flag_clear	clear CMP interrupt flag

#### Enum cmp\_enum

**Table 3-126. Enum cmp\_enum**

Member name	Function description
CMP0	comparator 0

#### cmp\_deinit

The description of cmp\_deinit is shown as below:

**Table 3-127. Function cmp\_deinit**

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

-	-
Return value	
-	-

Example:

```
/* deinitialize CMP */
```

```
cmp_deinit();
```

## cmp\_mode\_init

The description of cmp\_mode\_init is shown as below:

**Table 3-128. Function cmp\_mode\_init**

<b>Function name</b>	cmp_mode_init
<b>Function prototype</b>	void cmp_mode_init(cmp_enum cmp_periph, uint32_t inverting_input, uint32_t output_hysteresis);
<b>Function descriptions</b>	CMP mode init
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>inverting_input</b>	inverting input
CMP_INVERTING_INP UT_1_4VREFINT	VREFINT *1/4 input
CMP_INVERTING_INP UT_1_2VREFINT	VREFINT *1/2 input
CMP_INVERTING_INP UT_3_4VREFINT	VREFINT *3/4 input
CMP_INVERTING_INP UT_VREFINT	VREFINT input
CMP_INVERTING_INP UT_PA4	CMP inverting input PA4
CMP_INVERTING_INP UT_PA5	CMP inverting input PA5
CMP_INVERTING_INP UT_PA2	CMP inverting input PA2
CMP_INVERTING_INP UT_DAC0_OUT0	CMP inverting input DAC0_OUT0
<b>Input parameter{in}</b>	
<b>output_hysteresis</b>	hysteresis level
CMP_HYSTERESIS_N 0	output no hysteresis
CMP_HYSTERESIS_L	output low hysteresis

<i>OW</i>	
<i>CMP_HYSTERESIS_MIDDLE</i>	output middle hysteresis
<i>CMP_HYSTERESIS_HIGH</i>	output high hysteresis
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize CMP0 mode */
```

```
cmp_mode_init(CMP0, CMP_INVERTING_INPUT_1_4VREFINT, CMP_HYSTERESIS_NO);
```

### cmp\_output\_init

The description of cmp\_output\_init is shown as below:

**Table 3-129. Function cmp\_output\_init**

<b>Function name</b>	cmp_output_init
<b>Function prototype</b>	void cmp_output_init(cmp_enum cmp_periph, uint32_t output_polarity);
<b>Function descriptions</b>	CMP output init
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>output_polarity</b>	CMP output polarity
<i>CMP_OUTPUT_POLARITY_INVERTED</i>	output is inverted
<i>CMP_OUTPUT_POLARITY_NONINVERTED</i>	output is not inverted
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize CMP0 output */
```

```
cmp_output_init(CMP0, CMP_OUTPUT_POLARITY_NONINVERTED);
```

## cmp\_blanking\_init

The description of cmp\_blanking\_init is shown as below:

**Table 3-130. Function cmp\_blanking\_init**

<b>Function name</b>	cmp_blanking_init
<b>Function prototype</b>	void cmp_blanking_init(cmp_enum cmp_periph, uint32_t blanking_source_selection);
<b>Function descriptions</b>	CMP output blanking function init
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>blanking_source_selection</b>	blanking source selection
<b>CMP_BLANKING_NONE</b>	CMP no blanking source
<b>CMP_BLANKING_TIMER0_OC0</b>	CMP TIMER0_CH0 output compare signal selected as blanking source
<b>CMP_BLANKING_TIMER1_OC2</b>	CMP TIMER1_CH2 output compare signal selected as blanking source
<b>CMP_BLANKING_TIMER2_OC2</b>	CMP TIMER2_CH2 output compare signal selected as blanking source
<b>CMP_BLANKING_TIMER2_OC3</b>	CMP TIMER2_CH3 output compare signal selected as blanking source
<b>CMP_BLANKING_TIMER7_OC0</b>	CMP TIMER7_CH0 output compare signal selected as blanking source
<b>CMP_BLANKING_TIMER15_OC0</b>	CMP TIMER15_CH0 output compare signal selected as blanking source
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize CMP0 blanking function */
cmp_blanking_init(CMP0, CMP_BLANKING_NONE);
```

## cmp\_digital\_filter\_init

The description of cmp\_digital\_filter\_init is shown as below:

**Table 3-131. Function cmp\_digital\_filter\_init**

<b>Function name</b>	cmp_digital_filter_init
----------------------	-------------------------



<b>Function prototype</b>	void cmp_digital_filter_init(cmp_enum cmp_periph, uint32_t sample_frequency_selection, uint32_t filter_mode_selection);
<b>Function descriptions</b>	CMP digital filter init
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>sample_frequency_selection</b>	sampling frequency
<b>CMP_NOISE_FILTER_NONE</b>	CMP no noise filter
<b>CMP_SAMPLING_FREQUENCY_PCLK_DIV8</b>	digital filter is used and sampling frequency is PCLK/8
<b>CMP_SAMPLING_FREQUENCY_PCLK_DIV16</b>	digital filter is used and sampling frequency is PCLK/16
<b>CMP_SAMPLING_FREQUENCY_PCLK_DIV32</b>	digital filter is used and sampling frequency is PCLK/32
<b>Input parameter{in}</b>	
<b>Filter_mode_selection</b>	filter mode selection
<b>CMP_FILTER_MODE_3_TIMES</b>	output changes when the same value is sampled 3 times
<b>CMP_FILTER_MODE_4_TIMES</b>	output changes when the same value is sampled 4 times
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize CMP0 digital filter */
```

```
cmp_digital_filter_init(CMP0, CMP_SAMPLING_FREQUENCY_PCLK_DIV8, CMP_FILTER_MODE_3_TIMES);
```

## cmp\_enable

The description of cmp\_enable is shown as below:

**Table 3-132. Function cmp\_enable**

<b>Function name</b>	cmp_enable
<b>Function prototype</b>	void cmp_enable(cmp_enum cmp_periph);
<b>Function descriptions</b>	enable CMP

Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable CMP0 */
cmp_enable(CMP0);
```

### cmp\_disable

The description of cmp\_disable is shown as below:

**Table 3-133. Function cmp\_disable**

Function name	cmp_disable
Function prototype	void cmp_disable(cmp_enum cmp_periph);
Function descriptions	disable CMP
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable CMP0 */
cmp_disable(CMP0);
```

### cmp\_lock\_enable

The description of cmp\_lock\_enable is shown as below:

**Table 3-134. Function cmp\_lock\_enable**

Function name	cmp_lock_enable
Function prototype	void cmp_lock_enable(cmp_enum cmp_periph);
Function descriptions	lock the comparator
Precondition	-
The called functions	-

Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock CMP0 register */
cmp_lock_enable(CMP0);
```

### cmp\_voltage\_scaler\_enable

The description of cmp\_voltage\_scaler\_enable is shown as below:

**Table 3-135. Function cmp\_voltage\_scaler\_enable**

Function name	cmp_voltage_scaler_enable
Function prototype	void cmp_voltage_scaler_enable(cmp_enum cmp_periph);
Function descriptions	enable the voltage scaler
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable CMP0 the voltage scaler */
cmp_voltage_scaler_enable(CMP0);
```

### cmp\_voltage\_scaler\_disable

The description of cmp\_voltage\_scaler\_disable is shown as below:

**Table 3-136. Function cmp\_voltage\_scaler\_disable**

Function name	cmp_voltage_scaler_disable
Function prototype	void cmp_voltage_scaler_disable(cmp_enum cmp_periph);
Function descriptions	disable comparator the voltage scaler
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable CMP0 the voltage scaler */
cmp_voltage_scaler_disable(CMP0);
```

### cmp\_scaler\_bridge\_enable

The description of cmp\_scaler\_bridge\_enable is shown as below:

**Table 3-137. Function cmp\_scaler\_bridge\_enable**

Function name	cmp_scaler_bridge_enable
Function prototype	void cmp_scaler_bridge_enable(cmp_enum cmp_periph);
Function descriptions	enable the scaler bridge
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable CMP0 the scaler bridge */
cmp_scaler_bridge_enable(CMP0);
```

### cmp\_scaler\_bridge\_disable

The description of cmp\_scaler\_bridge\_disable is shown as below:

**Table 3-138. Function cmp\_scaler\_bridge\_disable**

Function name	cmp_scaler_bridge_disable
Function prototype	void cmp_scaler_bridge_disable(cmp_enum cmp_periph);
Function descriptions	disable the scaler bridge
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable CMP0 the scaler bridge */
```

```
cmp_scaler_bridge_disable(CMP0);
```

### cmp\_output\_level\_get

The description of cmp\_output\_level\_get is shown as below:

**Table 3-139. Function cmp\_output\_level\_get**

Function name	cmp_output_level_get
Function prototype	uint32_t cmp_output_level_get(cmp_enum cmp_periph);
Function descriptions	get output level
Precondition	-
The called functions	-
Input parameter{in}	
cmp_periph	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
Output parameter{out}	
-	-
Return value	
uint32_t	the output level
CMP_OUTPUTLEVEL_HIGH	comparator output high
CMP_OUTPUTLEVEL_LOW	comparator output low

Example:

```
uint32_t level;
```

```
/* get CMP0 output level */
```

```
level = cmp_output_level_get(CMP0);
```

### cmp\_flag\_get

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

**Table 3-140. Function cmp\_flag\_get**

Function name	cmp_flag_get
Function prototype	FlagStatus cmp_flag_get(cmp_enum cmp_periph, uint32_t flag);
Function descriptions	get CMP flag
Precondition	-
The called functions	-
Input parameter{in}	

<b>cmp_periph</b>	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>flag</b>	CMP interrupt flag
<b>CMP_FLAG_COMPAR E</b>	CMP compare flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the CMP0 interrupt flag bit */
```

```
FlagStatus flag_value;
```

```
flag_value = cmp_flag_get(CMP0, CMP_FLAG_COMPARE);
```

### cmp\_flag\_clear

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

**Table 3-141. Function cmp\_flag\_clear**

<b>Function name</b>	cmp_flag_clear
<b>Function prototype</b>	void cmp_flag_clear(cmp_enum cmp_periph, uint32_t flag);
<b>Function descriptions</b>	clear CMP flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>flag</b>	CMP interrupt flag
<b>CMP_FLAG_COMPAR E</b>	CMP compare flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the CMP0 interrupt flag bit */
```

```
cmp_flag_clear(CMP0, CMP_FLAG_COMPARE);
```

### cmp\_interrupt\_enable

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

**Table 3-142. Function cmp\_interrupt\_enable**

<b>Function name</b>	cmp_interrupt_enable
<b>Function prototype</b>	void cmp_interrupt_enable(cmp_enum cmp_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable CMP interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>interrupt</b>	CMP interrupt
<b>CMP_INT_COMPARE</b>	CMP compare interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the CMP0 interrupt */
cmp_interrupt_enable(CMP0, CMP_INT_COMPARE);
```

### cmp\_interrupt\_disable

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

**Table 3-143. Function cmp\_interrupt\_disable**

<b>Function name</b>	cmp_interrupt_disable
<b>Function prototype</b>	void cmp_interrupt_disable(cmp_enum cmp_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable CMP interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>interrupt</b>	CMP interrupt
<b>CMP_INT_COMPARE</b>	CMP compare interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the CMP0 interrupt */
cmp_interrupt_disable(CMP0, CMP_INT_COMPARE);
```

## cmp\_interrupt\_flag\_get

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

**Table 3-144. Function cmp\_interrupt\_flag\_get**

<b>Function name</b>	cmp_interrupt_flag_get
<b>Function prototype</b>	FlagStatus cmp_interrupt_flag_get(cmp_enum cmp_periph, uint32_t flag);
<b>Function descriptions</b>	get CMP interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>flag</b>	CMP interrupt flag
<b>CMP_INT_FLAG_COMPARE</b>	CMP compare interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the CMP0 interrupt bit*/
```

```
FlagStatus flag_value;
```

```
flag_value = cmp_interrupt_flag_get(CMP0, CMP_INT_FLAG_COMPARE);
```

## cmp\_interrupt\_flag\_clear

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

**Table 3-145. Function cmp\_interrupt\_flag\_clear**

<b>Function name</b>	cmp_interrupt_flag_clear
<b>Function prototype</b>	void cmp_interrupt_flag_clear(cmp_enum cmp_periph, uint32_t flag);
<b>Function descriptions</b>	clear CMP interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmp_periph</b>	refer to enum <a href="#">Table 3-126. Enum cmp_enum</a>
<b>Input parameter{in}</b>	
<b>flag</b>	CMP interrupt flag
<b>CMP_INT_FLAG_COMPARE</b>	CMP compare interrupt flag
<b>Output parameter{out}</b>	
-	-



Return value	
-	-

Example:

```
/* clear the CMP0 interrupt bit*/
```

```
cmp_interrupt_flag_clear(CMP0, CMP_INT_FLAG_COMPARE);
```

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

### 3.7.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

**Table 3-146. CRC Registers**

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

### 3.7.2. Descriptions of Peripheral functions

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

**Table 3-147. CRC firmware function**

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

Function name	Function description
crc_block_data_calculate	CRC calculate a data array

## crc\_deinit

The description of crc\_deinit is shown as below:

**Table 3-148. Function crc\_deinit**

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

Example:

```
/* deinit CRC calculation unit */
crc_deinit();
```

## crc\_init\_data\_register\_write

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

**Table 3-149. Function crc\_init\_data\_register\_write**

Function name	crc_init_data_register_write
Function prototype	void crc_init_data_register_write(uint32_t init_data);
Function descriptions	write the initialization data register
Precondition	-
The called functions	-
Input parameter{in}	
uint32_t	specify 32-bit data
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write the initialization data register */
crc_init_data_register_write(0x11223344);
```

## crc\_data\_register\_read

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

**Table 3-150. Function crc\_data\_register\_read**

<b>Function name</b>	crc_data_register_read
<b>Function prototype</b>	uint32_t crc_data_register_read(void);
<b>Function descriptions</b>	read the data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	32-bit value of the data register(0-0xFFFFFFFF)

Example:

```
/* read the data register */
uint32_t cdc_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 3-151. Function crc\_free\_data\_register\_read**

<b>Function name</b>	crc_free_data_register_read
<b>Function prototype</b>	uint8_t crc_free_data_register_read(void);
<b>Function descriptions</b>	read the free data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-152. 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 the free data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>free_data</b>	specify 8-bit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## crc\_reverse\_output\_data\_disable

The description of crc\_reverse\_output\_data\_disable is shown as below:

**Table 3-153. Function crc\_reverse\_output\_data\_disable**

<b>Function name</b>	crc_reverse_output_data_disable
<b>Function prototype</b>	void crc_reverse_output_data_disable(void);
<b>Function descriptions</b>	disable the reverse operation of output data
<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 reverse operation of output data */
crc_reverse_output_data_disable();
```

## crc\_reverse\_output\_data\_enable

The description of crc\_reverse\_output\_data\_enable is shown as below:

**Table 3-154. Function `crc_reverse_output_data_enable`**

<b>Function name</b>	<code>crc_reverse_output_data_enable</code>
<b>Function prototype</b>	<code>void crc_reverse_output_data_enable(void);</code>
<b>Function descriptions</b>	enable the reverse operation of output data
<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 reverse operation of output data */
```

```
crc_reverse_output_data_enable();
```

### **`crc_input_data_reverse_config`**

The description of `crc_input_data_reverse_config` is shown as below:

**Table 3-155. Function `crc_input_data_reverse_config`**

<b>Function name</b>	<code>crc_input_data_reverse_config</code>
<b>Function prototype</b>	<code>void crc_input_data_reverse_config(uint32_t data_reverse);</code>
<b>Function descriptions</b>	configure the crc input data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data_reverse</b>	specify input data reverse function
<code>CRC_INPUT_DATA_NOT</code>	input data is not reversed
<code>CRC_INPUT_DATA_BYTE</code>	input data is reversed on 8 bits
<code>CRC_INPUT_DATA_HALFWORD</code>	input data is reversed on 16 bits
<code>CRC_INPUT_DATA_WORD</code>	input data is reversed on 32 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the crc input data */
```

```
crc_input_data_reverse_config(CRC_INPUT_DATA_WORD);
```

### crc\_data\_register\_reset

The description of `crc_data_register_reset` is shown as below:

**Table 3-156. Function `crc_data_register_reset`**

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

Example:

```
/* reset data register to the value of initialization data register */
```

```
crc_data_register_reset();
```

### crc\_polynomial\_size\_set

The description of `crc_polynomial_size_set` is shown as below:

**Table 3-157. Function `crc_polynomial_size_set`**

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

Example:

```

/* configure the CRC polynomial size*/

crc_polynomial_size_set(CRC_CTL_PS_7);

```

### crc\_polynomial\_set

The description of crc\_polynomial\_set is shown as below:

**Table 3-158. Function crc\_polynomial\_set**

<b>Function name</b>	crc_polynomial_set
<b>Function prototype</b>	void crc_polynomial_set(uint32_t poly)
<b>Function descriptions</b>	configure the CRC polynomial value function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>poly</b>	configurable polynomial value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure the CRC polynomial value */

crc_polynomial_set(0x11223344);

```

### crc\_single\_data\_calculate

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

**Table 3-159. Function crc\_single\_data\_calculate**

<b>Function name</b>	crc_single_data_calculate
<b>Function prototype</b>	uint32_t crc_single_data_calculate(uint32_t sdata, uint8_t data_format);
<b>Function descriptions</b>	CRC calculate single data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sdata</b>	specify 32-bit data
<b>Input parameter{in}</b>	
<b>data_format</b>	input data format
<i>INPUT_FORMAT_WORD</i>	input data in word format
<i>INPUT_FORMAT_HALFWORD</i>	input data in half-word format
<i>INPUT_FORMAT_BYTE</i>	input data in byte format

Output parameter{out}	
-	-
Return value	
uint32_t	32-bit CRC calculate value (0-0xFFFFFFFF)

Example:

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

```
uint32_t val = 0, valcrc = 0;
```

```
val = (uint32_t)0xabcd1234;
```

```
valcrc = crc_single_data_calculate(val, INPUT_FORMAT_WORD);
```

### crc\_block\_data\_calculate

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

**Table 3-160. Function crc\_block\_data\_calculate**

Function name	crc_block_data_calculate
Function prototype	uint32_t crc_block_data_calculate(void *array, uint32_t size, uint8_t data_format);
Function descriptions	CRC calculate a data array
Precondition	-
The called functions	-
Input parameter{in}	
array	array of the input data array
Input parameter{in}	
size	size of the array
Input parameter{in}	
data_format	input data format
INPUT_FORMAT_WORD	input data in word format
INPUT_FORMAT_HALFWORD	input data in half-word format
INPUT_FORMAT_BYTE	input data in byte format
Output parameter{out}	
-	-
Return value	
uint32_t	32-bit CRC calculate value (0-0xFFFFFFFF)

Example:

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

```
#define BUFFER_SIZE 6
```



```
uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

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

valcrc = crc_block_data_calculate(data_buffer, BUFFER_SIZE, INPUT_FORMAT_WORD);
```

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

### 3.8.1. Descriptions of Peripheral registers

CTC registers are listed in the table shown as below:

**Table 3-161. 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

### 3.8.2. Descriptions of Peripheral functions

CTC functions are listed in the table shown as below:

**Table 3-162. 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

Function name	Function description
	pulse occurred
<code>ctc_counter_direction_read</code>	read CTC trim counter direction when reference sync pulse occurred
<code>ctc_counter_reload_value_read</code>	read CTC counter reload value
<code>ctc_irc48m_trim_value_read</code>	read the IRC48M trim value
<code>ctc_interrupt_enable</code>	enable the CTC interrupt
<code>ctc_interrupt_disable</code>	disable the CTC interrupt
<code>ctc_interrupt_flag_get</code>	get CTC interrupt flag
<code>ctc_interrupt_flag_clear</code>	clear CTC interrupt flag
<code>ctc_flag_get</code>	get CTC flag
<code>ctc_flag_clear</code>	clear CTC flag

### **ctc\_deinit**

The description of `ctc_deinit` is shown as below:

**Table 3-163. Function `ctc_deinit`**

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

Example:

```
/* reset CTC */
```

```
ctc_deinit();
```

### **ctc\_counter\_enable**

The description of `ctc_counter_enable` is shown as below:

**Table 3-164. Function `ctc_counter_enable`**

Function name	<code>ctc_counter_enable</code>
Function prototype	<code>void ctc_counter_enable (void);</code>
Function descriptions	enable CTC counter
Precondition	-
The called functions	-

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

Example:

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

```
ctc_counter_enable ();
```

### ctc\_counter\_disable

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

**Table 3-165. Function ctc\_counter\_disable**

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

Example:

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

```
ctc_counter_disable ();
```

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

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

**Table 3-166. Function ctc\_irc48m\_trim\_value\_config**

Function name	ctc_irc48m_trim_value_config
Function prototype	void ctc_irc48m_trim_value_config(uint32_t trim_value);
Function descriptions	configure the IRC48M trim value
Precondition	-
The called functions	-
Input parameter{in}	
trim_value	0~63
Output parameter{out}	

-	-
Return value	
-	-

Example:

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

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

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

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

**Table 3-167. Function ctc\_software\_refsource\_pulse\_generate**

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

Example:

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

```
ctc_software_refsource_pulse_generate ();
```

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

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

**Table 3-168. Function ctc\_hardware\_trim\_mode\_config**

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

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

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 3-169. Function ctc\_refsource\_polarity\_config**

<b>Function name</b>	<b>ctc_refsource_polarity_config</b>
<b>Function prototype</b>	<b>void ctc_refsource_polarity_config(uint32_t polarity);</b>
<b>Function descriptions</b>	<b>configure reference signal source polarity</b>
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>polarity</b>	<b>reference signal source polarity</b>
<b>CTC_REFSOURCE_POLARITY_FALLING</b>	<b>reference signal source polarity is falling edge</b>
<b>CTC_REFSOURCE_POLARITY_RISING</b>	<b>reference signal source polarity is rising edge</b>
<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 3-170. Function ctc\_refsource\_signal\_select**

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

The called functions	-
Input parameter{in}	
refs	reference signal source
CTC_REFSOURCE_G PIO	GPIO is selected
CTC_REFSOURCE_L XTAL	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 3-171. Function ctc\_refsource\_prescaler\_config**

Function name	ctc_refsource_prescaler_config
Function prototype	void ctc_refsource_prescaler_config(uint32_t prescaler);
Function descriptions	configure reference signal source prescaler
Precondition	-
The called functions	-
Input parameter{in}	
prescaler	Prescaler factor
CTC_REFSOURCE_P SC_OFF	reference signal not divided
CTC_REFSOURCE_P SC_DIV2	reference signal divided by 2
CTC_REFSOURCE_P SC_DIV4	reference signal divided by 4
CTC_REFSOURCE_P SC_DIV8	reference signal divided by 8
CTC_REFSOURCE_P SC_DIV16	reference signal divided by 16
CTC_REFSOURCE_P SC_DIV32	reference signal divided by 32
CTC_REFSOURCE_P SC_DIV64	reference signal divided by 64
CTC_REFSOURCE_P SC_DIV128	reference signal divided by 128

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
ctc_refsource_prescaler_config(CTC_REFSOURCE_PSC_DIV2);
```

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

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

**Table 3-172. Function ctc\_clock\_limit\_value\_config**

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

Example:

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

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

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

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

**Table 3-173. Function ctc\_counter\_reload\_value\_config**

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

Return value	
-	-

Example:

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

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

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

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

**Table 3-174. Function ctc\_counter\_capture\_value\_read**

Function name	ctc_counter_capture_value_read
Function prototype	uint16_t ctc_counter_capture_value_read(void);
Function descriptions	read CTC counter capture value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	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 3-175. 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 3-176. Function ctc\_counter\_reload\_value\_read**

Function name	ctc_counter_reload_value_read
Function prototype	uint16_t ctc_counter_reload_value_read(void);
Function descriptions	read CTC counter reload value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	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 3-177. Function ctc\_irc48m\_trim\_value\_read**

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

Return value	
uint8_t	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 3-178. Function ctc\_interrupt\_enable**

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

Example:

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

ctc_interrupt_enable (CTC_INT_CKOK);
```

### ctc\_interrupt\_disable

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

**Table 3-179. Function ctc\_interrupt\_disable**

Function name	ctc_interrupt_disable
Function prototype	void ctc_interrupt_disable(uint32_t interrupt);
Function descriptions	disable the CTC interrupt
Precondition	-
The called functions	-
Input parameter{in}	

interrupt	CTC interrupt
CTC_INT_CKOK	clock trim OK interrupt
CTC_INT_CKWARN	clock trim warning interrupt
CTC_INT_ERR	error interrupt
CTC_INT_EREf	expect reference interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
ctc_interrupt_disable (CTC_INT_CKOK);
```

### ctc\_interrupt\_flag\_get

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

**Table 3-180. Function ctc\_interrupt\_flag\_get**

Function name	ctc_interrupt_flag_get
Function prototype	FlagStatus ctc_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get CTC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	CTC interrupt flag
CTC_INT_FLAG_CKOK	clock trim OK interrupt
CTC_INT_FLAG_CKWARN	clock trim warning interrupt
CTC_INT_FLAG_ERR	error interrupt
CTC_INT_FLAG_EREf	expect reference interrupt
CTC_INT_FLAG_CKErr	clock trim error bit interrupt
CTC_INT_FLAG_REFMISS	reference sync pulse miss interrupt
CTC_INT_FLAG_TRIMERR	trim value error interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

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

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

### ctc\_interrupt\_flag\_clear

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

**Table 3-181. Function ctc\_interrupt\_flag\_clear**

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

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 3-182. Function ctc\_flag\_get**

Function name	ctc_flag_get
---------------	--------------

Function prototype	FlagStatus ctc_flag_get(uint32_t flag);
Function descriptions	get CTC status flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	CTC status flag
CTC_FLAG_CKOK	clock trim OK interrupt flag
CTC_FLAG_CKWARN	clock trim warning interrupt flag
CTC_FLAG_ERR	error interrupt flag
CTC_FLAG_EREf	expect reference interrupt flag
CTC_FLAG_CKERR	clock trim error bit
CTC_FLAG_REFMISS	reference sync pulse miss flag
CTC_FLAG_TRIMERR	trim value error flag
Output parameter{out}	
-	-
Return value	
FlagStatus	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 3-183. Function ctc\_flag\_clear**

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

Return value	
-	-

Example:

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

```
ctc_flag_clear (CTC_FLAG_CKOK);
```

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

### 3.9.1. Peripheral register description

DAC registers are listed in the table shown as below:

**Table 3-184. 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_OUT0_DO	DACx_OUT0 data output register
<b>DAC_STAT0</b>	DACx_OUT0 status register 0

### 3.9.2. Descriptions of Peripheral functions

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

**Table 3-185. DAC firmware functions**

Function name	Function description
dac_deinit	deinitialize DAC
dac_enable	enable DAC
dac_disable	disable DAC
dac_dma_enable	enable DAC DMA function
dac_dma_disable	disable DAC DMA function
<b>dac_pin_select</b>	DAC output pin selection
dac_output_buffer_enable	enable DAC output buffer
dac_output_buffer_disable	disable DAC output buffer
dac_output_value_get	get DAC output value

Function name	Function description
<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_output_connect_to_pin_enable</code>	enable DAC output connect to pin
<code>dac_output_connect_to_pin_disable</code>	disable DAC output connect to pin
<code>dac_flag_get</code>	get the DAC flag
<code>dac_flag_clear</code>	clear the DAC flag
<code>dac_interrupt_enable</code>	enable DAC interrupt
<code>dac_interrupt_disable</code>	disable DAC interrupt
<code>dac_interrupt_flag_get</code>	get the DAC interrupt flag
<code>dac_interrupt_flag_clear</code>	clear the DAC interrupt flag

### **dac\_deinit**

The description of `dac_deinit` is shown as below:

**Table 3-186. Function `dac_deinit`**

<b>Function name</b>	<code>dac_deinit</code>
<b>Function prototype</b>	<code>void dac_deinit(uint32_t dac_periph);</code>
<b>Function descriptions</b>	deinitialize DAC
<b>Precondition</b>	-
<b>The called functions</b>	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
<b>Input parameter{in}</b>	
<b><code>dac_periph</code></b>	DAC peripheral
<b><code>DACx</code></b>	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 3-187. Function `dac_enable`

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

Example:

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

### `dac_disable`

The description of `dac_disable` is shown as below:

Table 3-188. Function `dac_disable`

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

Example:

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



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

## **dac\_dma\_enable**

The description of `dac_dma_enable` is shown as below:

**Table 3-189. 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)
<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 3-190. Function `dac_dma_disable`**

<b>Function name</b>	<code>dac_dma_disable</code>
<b>Function prototype</b>	<code>void dac_dma_disable(uint32_t dac_periph, uint8_t dac_out);</code>
<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)
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

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

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

### **dac\_pin\_select**

The description of `dac_pin_sel` is shown as below:

**Table 3-191. Function `dac_dma_disable`**

Function name	<code>dac_pin_select</code>
Function prototype	<code>void dac_pin_select(uint32_t dac_periph, uint8_t dac_out, uint32_t pin_sel);</code>
Function descriptions	DAC output pin selection
Precondition	-
The called functions	-
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)
Input parameter{in}	
<b>pin_sel</b>	pin selection
<i>DAC_OUTPUT_CONNECT_PA4</i>	DAC output pin connect PA4
<i>DAC_OUTPUT_CONNECT_PA5</i>	DAC output pin connect PA5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DAC output pin select PA4 */
```

```
dac_pin_select (DAC0, DAC_OUT0, DAC_OUTPUT_CONNECT_PA4);
```

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

The description of `dac_output_buffer_enable` is shown as below:

Table 3-192. Function `dac_output_buffer_enable`

<b>Function name</b>	<code>dac_output_buffer_enable</code>
<b>Function prototype</b>	<code>void dac_output_buffer_enable(uint32_t dac_periph, uint8_t dac_out);</code>
<b>Function descriptions</b>	enable DAC output buffer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b><code>dac_periph</code></b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b><code>dac_out</code></b>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0)
<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 3-193. Function `dac_output_buffer_disable`

<b>Function name</b>	<code>dac_output_buffer_disable</code>
<b>Function prototype</b>	<code>void dac_output_buffer_disable(uint32_t dac_periph, uint8_t dac_out);</code>
<b>Function descriptions</b>	disable DAC output buffer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b><code>dac_periph</code></b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b><code>dac_out</code></b>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0)
<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 3-194. Function `dac_output_value_get`**

<b>Function name</b>	<code>dac_output_value_get</code>
<b>Function prototype</b>	<code>uint16_t dac_output_value_get(uint32_t dac_periph);</code>
<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)
<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 3-195. 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)
<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 3-196. Function `dac_trigger_enable`**

<b>Function name</b>	<code>dac_trigger_enable</code>
<b>Function prototype</b>	<code>void dac_trigger_enable(uint32_t dac_periph, uint8_t dac_out);</code>
<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)
<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 3-197. 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)
<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 3-198. Function dac\_trigger\_source\_config**

<b>Function name</b>	dac_trigger_source_config
<b>Function prototype</b>	void dac_trigger_source_config(uint32_t dac_periph, uint8_t dac_out, uint32_t triggersource);
<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)
<b>Input parameter{in}</b>	
<b>triggersource</b>	external trigger of DAC

<i>DAC_TRIGGER_EXTERNAL</i>	external trigger selected from TRIGSEL
<i>DAC_TRIGGER_SOFTWARE</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 3-199. 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)
<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 3-200. Function `dac_wave_mode_config`

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

Example:

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

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

### `dac_lfsr_noise_config`

The description of `dac_lfsr_noise_config` is shown as below:

Table 3-201. Function `dac_lfsr_noise_config`

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



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

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 3-202. 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>	-
<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)
<b>Input parameter{in}</b>	
<b>amplitude</b>	the amplitude of the triangle
<i>DAC_TRIANGLE_AMPLITUDE_x</i>	$x = 2^n - 1$ (n = 1..12)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## dac\_output\_connect\_to\_pin\_enable

The description of dac\_output\_connect\_to\_pin\_enable is shown as below:

**Table 3-203. Function dac\_output\_connect\_to\_pin\_enable**

<b>Function name</b>	dac_output_connect_to_pin_enable
<b>Function prototype</b>	void dac_output_connect_to_pin_enable(uint32_t dac_periph, uint8_t dac_out);
<b>Function descriptions</b>	enable DAC output connect to pin
<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)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DAC0 output connect to pin */
```

```
dac_output_connect_to_pin_enable (DAC0, DAC_OUT0);
```

## dac\_output\_connect\_to\_pin\_disable

The description of dac\_output\_connect\_to\_pin\_disable is shown as below:

**Table 3-204. Function dac\_output\_connect\_to\_pin\_disable**

<b>Function name</b>	dac_output_connect_to_pin_disable
<b>Function prototype</b>	void dac_output_connect_to_pin_disable(uint32_t dac_periph, uint8_t dac_out);
<b>Function descriptions</b>	disable DAC output connect to pin
<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)
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* disable DAC0 output connect to pin */
```

```
dac_output_connect_to_pin_disable (DAC0, DAC_OUT0);
```

### **dac\_flag\_get**

The description of `dac_flag_get` is shown as below:

**Table 3-205. Function `dac_flag_get`**

<b>Function name</b>	<code>dac_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus dac_flag_get(uint32_t dac_periph, uint32_t flag);</code>
<b>Function descriptions</b>	get DAC flag
<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 = 0,1,2,3)
<b>Input parameter{in}</b>	
<b>flag</b>	the DAC status flags
<i>DAC_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	the state of DAC bit (SET or RESET)

Example:

```
/* get DAC0 flag */
```

```
FlagStatus flag;
```

```
flag = dac_flag_get(DAC0, DAC_FLAG_DDUDR0);
```

### **dac\_flag\_clear**

The description of `dac_flag_clear` is shown as below:

**Table 3-206. Function `dac_flag_clear`**

<b>Function name</b>	<code>dac_flag_clear</code>
<b>Function prototype</b>	<code>void dac_flag_clear(uint32_t dac_periph, uint32_t flag);</code>
<b>Function descriptions</b>	clear DAC flag

<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 = 0,1,2,3)
<b>Input parameter{in}</b>	
<b>flag</b>	DAC flag
<i>DAC_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear DAC0 flag */
```

```
dac_flag_clear(DAC0, DAC_FLAG_DDUDR0);
```

### **dac\_interrupt\_enable**

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

**Table 3-207. Function dac\_interrupt\_enable**

<b>Function name</b>	dac_interrupt_enable
<b>Function prototype</b>	void dac_interrupt_enable(uint32_t dac_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable DAC interrupt
<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 = 0,1,2,3)
<b>Input parameter{in}</b>	
<b>interrupt</b>	the DAC interrupt
<i>DAC_INT_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DAC0 interrupt */
```

```
dac_interrupt_enable (DAC0, DAC_INT_DDUDR0);
```

## dac\_interrupt\_disable

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

**Table 3-208. Function dac\_interrupt\_disable**

<b>Function name</b>	dac_interrupt_disable
<b>Function prototype</b>	void dac_interrupt_disable(uint32_t dac_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable DAC interrupt
<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 = 0,1,2,3)
<b>Input parameter{in}</b>	
<b>interrupt</b>	the DAC interrupt
<i>DAC_INT_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DAC0 interrupt */
```

```
dac_interrupt_disable (DAC0, DAC_INT_DDUDR0);
```

## dac\_interrupt\_flag\_get

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

**Table 3-209. Function dac\_interrupt\_flag\_get**

<b>Function name</b>	dac_interrupt_flag_get
<b>Function prototype</b>	FlagStatus dac_interrupt_flag_get(uint32_t dac_periph, uint32_t int_flag);
<b>Function descriptions</b>	get DAC interrupt flag
<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 = 0,1,2,3)
<b>Input parameter{in}</b>	
<b>int_flag</b>	DAC interrupt flag
<i>DAC_INT_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt flag
<b>Output parameter{out}</b>	
-	-

Return value	
<b>FlagStatus</b>	the state of DAC interrupt flag(SET or RESET)

Example:

```
/* get DAC0 interrupt flag */
```

```
FlagStatus flag;
```

```
flag = dac_interrupt_flag_get(DAC0, DAC_INT_FLAG_DDUDR0);
```

### **dac\_interrupt\_flag\_clear**

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

**Table 3-210. Function dac\_interrupt\_flag\_clear**

<b>Function name</b>	dac_interrupt_flag_clear
<b>Function prototype</b>	void dac_interrupt_flag_clear(uint32_t dac_periph, uint32_t int_flag);
<b>Function descriptions</b>	clear DAC interrupt flag
<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 = 0,1,2,3)
<b>Input parameter{in}</b>	
<b>int_flag</b>	DAC interrupt flag
<i>DAC_INT_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear DAC0 interrupt flag */
```

```
dac_interrupt_flag_clear(DAC0, DAC_INT_FLAG_DDUDR0);
```

## **3.10. DBG**

The debugging system assists the debugger in debugging peripherals or performing debugging in low-power modes. The DBG registers are listed in chapter [3.10.1](#). the DBG firmware functions are introduced in chapter [3.10.2](#).

### **3.10.1. Descriptions of Peripheral registers**

DBG registers are listed in the table shown as below:

**Table 3-211. DBG Registers**

Registers	Descriptions
DBG_ID	DBG_ID code register
DBG_CTL	DBG control register

### 3.10.2. Descriptions of Peripheral functions

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

**Table 3-212. DBG firmware function**

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

### Enum dbg\_periph\_enum

**Table 3-213. Enum dbg\_periph\_enum**

Member name	Descriptions
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted
DBG_TIMER0_HOLD	hold TIMER0 counter when core is halted
DBG_TIMER1_HOLD	hold TIMER1 counter when core is halted
DBG_TIMER2_HOLD	hold TIMER2 counter when core is halted
DBG_TIMER3_HOLD	hold TIMER3 counter when core is halted
DBG_CAN0_HOLD	debug CAN0 kept when core is halted
DBG_I2C0_HOLD	hold I2C0 smbus when core is halted
DBG_I2C1_HOLD	hold I2C1 smbus when core is halted
DBG_TIMER7_HOLD	hold TIMER7 counter when core is halted
DBG_TIMER4_HOLD	hold TIMER4 counter when core is halted
DBG_TIMER5_HOLD	hold TIMER5 counter when core is halted
DBG_TIMER6_HOLD	hold TIMER6 counter when core is halted
DBG_CAN1_HOLD	debug CAN1 kept when core is halted
DBG_TIMER15_HOLD	hold TIMER15 counter when core is halted
DBG_TIMER16_HOLD	hold TIMER16 counter when core is halted

## dbg\_deinit

The description of dbg\_deinit is shown as below:

**Table 3-214. Function dbg\_deinit**

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

Example:

```
/* reset DBG */
dbg_deinit();
```

## dbg\_id\_get

The description of dbg\_id\_get is shown as below:

**Table 3-215. Function dbg\_id\_get**

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

Example:

```
/* read DBG_ID code */
uint32_t id = 0;
id = dbg_id_get();
```



## dbg\_low\_power\_enable

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

**Table 3-216. 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_STANDBY);
```

## dbg\_low\_power\_disable

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

**Table 3-217. 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>	Do not keep debugger connection during sleep mode
<i>DBG_LOW_POWER_DEEPSLEEP</i>	Do not keep debugger connection during deepsleep mode
<i>DBG_LOW_POWER_STANDBY</i>	Do not keep debugger connection during standby mode

<i>TANDBY</i>	
<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_STANDBY);
```

### dbg\_periph\_enable

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

**Table 3-218. 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>	DBG peripheral
<i>DBG_FWDGT_HOLD</i>	debug FWDGT kept when core is halted
<i>DBG_WWDGT_HOLD</i>	debug WWDGT kept when core is halted
<i>DBG_CANx_HOLD</i>	hold CANx (x=0,1) counter when core is halted
<i>DBG_I2Cx_HOLD</i>	hold I2Cx (x=0,1) smbus when core is halted
<i>DBG_TIMERx_HOLD</i>	hold TIMERx (x=0,1,2,3,4,5,6,7,15,16) 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_TIMER1_HOLD);
```

### dbg\_periph\_disable

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

**Table 3-219. 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>	DBG peripheral
<i>DBG_FWDGT_HOLD</i>	debug FWDGT kept when core is halted
<i>DBG_WWDGT_HOLD</i>	debug WWDGT kept when core is halted
<i>DBG_CANx_HOLD</i>	hold CANx (x=0,1) counter when core is halted
<i>DBG_I2Cx_HOLD</i>	hold I2Cx (x=0,1) smbus when core is halted
<i>DBG_TIMERx_HOLD</i>	hold TIMERx (x=0,1,2,3,4,5,6,7,15,16) 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_TIMER1_HOLD);
```

### dbg\_trace\_pin\_enable

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

**Table 3-220. 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 3-221. 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();
```

### dbg\_trace\_pin\_mode\_set

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

**Table 3-222. Function dbg\_trace\_pin\_mode\_set**

<b>Function name</b>	dbg_trace_pin_mode_set
<b>Function prototype</b>	void dbg_trace_pin_mode_set(uint32_t trace_mode)
<b>Function descriptions</b>	set trace pin mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>trace_mode</b>	trace pin mode
TRACE_MODE_ASYNC	trace pin used for async mode
TRACE_MODE_SYNC_DATASIZE_1	trace pin used for sync mode and data size is 1
TRACE_MODE_SYNC_DATASIZE_2	trace pin used for sync mode and data size is 2
TRACE_MODE_SYNC_DATASIZE_4	trace pin used for sync mode and data size is 4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* trace pin used for async mode */
```

```
dbg_trace_pin_mode_set(TRACE_MODE_ASYNC);
```

## 3.11. DMA / DMAMUX

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 [3.11.1](#), the DMA firmware functions are introduced in chapter [3.11.2](#)

DMAMUX is a transmission scheduler for DMA requests. The DMAMUX request multiplexer is used for routing a DMA request line between the peripherals / generated DMA request (from the DMAMUX request generator) and the DMA controller. The DMAMUX registers are listed in chapter [3.11.1](#), the DMAMUX firmware functions are introduced in chapter [3.11.2](#)

### 3.11.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

**Table 3-223. DMA Registers**

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

DMAMUX registers are listed in the table shown as below:

**Table 3-224. DMAMUX Registers**

Registers	Descriptions
DMAMUX_RM_CHx CFG (x=0..11)	Request multiplexer channel x configuration register
DMAMUX_RM_INT F	Request multiplexer channel interrupt flag register
DMAMUX_RM_INT C	Request multiplexer channel interrupt flag clear register
DMAMUX_RG_CHx CFG (x=0..3)	Request generator channel x configuration register
DMAMUX_RG_INT F	Request generator channel interrupt flag register

Registers	Descriptions
DMAMUX_RG_INT C	Rquest generator channel interrupt flag clear register

### 3.11.2. Descriptions of Peripheral functions

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

**Table 3-225. DMA firmware function**

Function name	Function description
dma_deinit	deinitialize DMA a channel registers
dma_struct_para_init	initialize the parameters of DMA struct with the default values
dma_init	initialize DMA channel
dma_circulation_enable	enable DMA circulation mode
dma_circulation_disable	disable DMA circulation mode
dma_memory_to_memory_enable	enable memory to memory mode
dma_memory_to_memory_disable	disable memory to memory mode
dma_channel_enable	enable DMA channel
dma_channel_disable	disable DMA channel
dma_periph_address_config	set DMA peripheral base address
dma_memory_address_config	set DMA memory base address
dma_transfer_number_config	set the number of remaining data to be transferred by the DMA
dma_transfer_number_get	get the number of remaining data to be transferred by the DMA
dma_priority_config	configure priority level of DMA channel
dma_memory_width_config	configure transfer data size of memory
dma_periph_width_config	configure transfer data size of peripheral
dma_memory_increase_enable	enable next address increasement algorithm of memory
dma_memory_increase_disable	disable next address increasement algorithm of memory
dma_periph_increase_enable	enable next address increasement algorithm of peripheral
dma_periph_increase_disable	disable next address increasement algorithm of peripheral
dma_transfer_direction_config	configure the direction of data transfer on the channel
dma_flag_get	check DMA flag is set or not
dma_flag_clear	clear DMA a channel flag
dma_interrupt_enable	enable DMA interrupt
dma_interrupt_disable	disable DMA interrupt
dma_interrupt_flag_get	check DMA flag and interrupt enable bit is set or not
dma_interrupt_flag_clear	clear DMA a channel flag

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

**Table 3-226. DMAMUX firmware function**

Function name	Function description
dmamux_sync_struct_para_init	initialize the parameters of DMAMUX synchronization mode

Function name	Function description
	structure with the default values
dmamux_synchronization_init	initialize DMAMUX request multiplexer channel synchronization mode
dmamux_synchronization_enable	enable synchronization mode
dmamux_synchronization_disable	disable synchronization mode
dmamux_event_generation_enable	enable event generation
dmamux_event_generation_disable	disable event generation
dmamux_gen_struct_para_init	initialize the parameters of DMAMUX request generator structure with the default values
dmamux_request_generator_init	initialize DMAMUX request generator channel
dmamux_request_generator_channel_enable	enable DMAMUX request generator channel
dmamux_request_generator_channel_disable	disable DMAMUX request generator channel
dmamux_synchronization_polarity_config	configure synchronization input polarity
dmamux_request_forward_number_config	configure number of DMA requests to forward
dmamux_sync_id_config	configure synchronization input identification
dmamux_request_id_config	configure multiplexer input identification
dmamux_trigger_polarity_config	configure trigger input polarity
dmamux_request_generate_number_config	configure number of DMA requests to be generated
dmamux_trigger_id_config	configure trigger input identification
dmamux_flag_get	get DMAMUX flag
dmamux_flag_clear	clear DMAMUX flag
dmamux_interrupt_enable	enable DMAMUX interrupt
dmamux_interrupt_disable	disable DMAMUX interrupt
dmamux_interrupt_flag_get	get DMAMUX interrupt flag
dmamux_interrupt_flag_clear	clear DMAMUX interrupt flag

## Structure dma\_parameter\_struct

**Table 3-227. Structure dma\_parameter\_struct**

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

direction	channel data transfer direction
request	channel input identification
circular_mode	Circular mode

### Structure dmamux\_sync\_parameter\_struct

**Table 3-228. Structure dmamux\_sync\_parameter\_struct**

Member name	Function description
sync_id	synchronization input identification
sync_polarity	synchronization input polarity
request_number	number of DMA requests to forward

### Structure dmamux\_gen\_parameter\_struct

**Table 3-229. Structure dmamux\_gen\_parameter\_struct**

Member name	Function description
trigger_id	trigger input identification
trigger_polarity	DMAMUX request generator trigger polarity
request_number	number of DMA requests to be generated

### Enum dma\_channel\_enum

**Table 3-230. Enum dma\_channel\_enum**

Member name	Function description
DMA_CH0	DMA Channel 0
DMA_CH1	DMA Channel 1
DMA_CH2	DMA Channel 2
DMA_CH3	DMA Channel 3
DMA_CH4	DMA Channel 4
DMA_CH5	DMA Channel 5
DMA_CH6	DMA Channel 6

### Enum dmamux\_multiplexer\_channel\_enum

**Table 3-231. Enum dmamux\_multiplexer\_channel\_enum**

Member name	Function description
DMAMUX_MUXCH 0	DMAMUX request multiplexer Channel 0
DMAMUX_MUXCH 1	DMAMUX request multiplexer Channel 1
DMAMUX_MUXCH 2	DMAMUX request multiplexer Channel 2
DMAMUX_MUXCH 3	DMAMUX request multiplexer Channel 3
DMAMUX_MUXCH	DMAMUX request multiplexer Channel 4



4	
DMAMUX_MUXCH 5	DMAMUX request multiplexer Channel 5
DMAMUX_MUXCH 6	DMAMUX request multiplexer Channel 6
DMAMUX_MUXCH 7	DMAMUX request multiplexer Channel 7
DMAMUX_MUXCH 8	DMAMUX request multiplexer Channel 8
DMAMUX_MUXCH 9	DMAMUX request multiplexer Channel 9
DMAMUX_MUXCH 10	DMAMUX request multiplexer Channel 10
DMAMUX_MUXCH 11	DMAMUX request multiplexer Channel 11

### Enum dmamux\_generator\_channel\_enum

**Table 3-232. Enum dmamux\_generator\_channel\_enum**

Member name	Function description
DMAMUX_GENCH0	DMAMUX request generator Channel0
DMAMUX_GENCH1	DMAMUX request generator Channel1
DMAMUX_GENCH2	DMAMUX request generator Channel2
DMAMUX_GENCH3	DMAMUX request generator Channel3

### Enum dmamux\_interrupt\_enum

**Table 3-233. Enum dmamux\_interrupt\_enum**

Member name	Function description
DMAMUX_INT_MU XCH0_SO	DMAMUX request multiplexer channel 0 synchronization overrun interrupt
DMAMUX_INT_MU XCH1_SO	DMAMUX request multiplexer channel 1 synchronization overrun interrupt
DMAMUX_INT_MU XCH2_SO	DMAMUX request multiplexer channel 2 synchronization overrun interrupt
DMAMUX_INT_MU XCH3_SO	DMAMUX request multiplexer channel 3 synchronization overrun interrupt
DMAMUX_INT_MU XCH4_SO	DMAMUX request multiplexer channel 4 synchronization overrun interrupt
DMAMUX_INT_MU XCH5_SO	DMAMUX request multiplexer channel 5 synchronization overrun interrupt
DMAMUX_INT_MU XCH6_SO	DMAMUX request multiplexer channel 6 synchronization overrun interrupt
DMAMUX_INT_MU	DMAMUX request multiplexer channel 7 synchronization overrun interrupt

XCH7_SO	
DMAMUX_INT_MU XCH8_SO	DMAMUX request multiplexer channel 8 synchronization overrun interrupt
DMAMUX_INT_MU XCH9_SO	DMAMUX request multiplexer channel 9 synchronization overrun interrupt
DMAMUX_INT_MU XCH10_SO	DMAMUX request multiplexer channel 10 synchronization overrun interrupt
DMAMUX_INT_MU XCH11_SO	DMAMUX request multiplexer channel 11 synchronization overrun interrupt
DMAMUX_INT_GE NCH0_TO	DMAMUX request generator channel 0 trigger overrun interrupt
DMAMUX_INT_GE NCH1_TO	DMAMUX request generator channel 1 trigger overrun interrupt
DMAMUX_INT_GE NCH2_TO	DMAMUX request generator channel 2 trigger overrun interrupt
DMAMUX_INT_GE NCH3_TO	DMAMUX request generator channel 3 trigger overrun interrupt

### Enum dmamux\_flag\_enum

**Table 3-234. Enum dmamux\_flag\_enum**

Member name	Function description
DMAMUX_FLAG_M UXCH0_SO	DMAMUX request multiplexer channel 0 synchronization overrun flag
DMAMUX_FLAG_M UXCH1_SO	DMAMUX request multiplexer channel 1 synchronization overrun flag
DMAMUX_FLAG_M UXCH2_SO	DMAMUX request multiplexer channel 2 synchronization overrun flag
DMAMUX_FLAG_M UXCH3_SO	DMAMUX request multiplexer channel 3 synchronization overrun flag
DMAMUX_FLAG_M UXCH4_SO	DMAMUX request multiplexer channel 4 synchronization overrun flag
DMAMUX_FLAG_M UXCH5_SO	DMAMUX request multiplexer channel 5 synchronization overrun flag
DMAMUX_FLAG_M UXCH6_SO	DMAMUX request multiplexer channel 6 synchronization overrun flag
DMAMUX_FLAG_M UXCH7_SO	DMAMUX request multiplexer channel 7 synchronization overrun flag
DMAMUX_FLAG_M UXCH8_SO	DMAMUX request multiplexer channel 8 synchronization overrun flag
DMAMUX_FLAG_M UXCH9_SO	DMAMUX request multiplexer channel 9 synchronization overrun flag
DMAMUX_FLAG_M UXCH10_SO	DMAMUX request multiplexer channel 10 synchronization overrun flag

DMAMUX_FLAG_MUXCH11_SO	DMAMUX request multiplexer channel 11 synchronization overrun flag
DMAMUX_FLAG_GENCH0_TO	DMAMUX request generator channel 0 trigger overrun flag
DMAMUX_FLAG_GENCH1_TO	DMAMUX request generator channel 1 trigger overrun flag
DMAMUX_FLAG_GENCH2_TO	DMAMUX request generator channel 2 trigger overrun flag
DMAMUX_FLAG_GENCH3_TO	DMAMUX request generator channel 3 trigger overrun flag

### Enum dmamux\_interrupt\_flag\_enum

**Table 3-235. Enum dmamux\_interrupt\_flag\_enum**

Member name	Function description
DMAMUX_INT_FLAG_MUXCH0_SO	DMAMUX request multiplexer channel 0 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH1_SO	DMAMUX request multiplexer channel 1 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH2_SO	DMAMUX request multiplexer channel 2 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH3_SO	DMAMUX request multiplexer channel 3 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH4_SO	DMAMUX request multiplexer channel 4 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH5_SO	DMAMUX request multiplexer channel 5 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH6_SO	DMAMUX request multiplexer channel 6 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH7_SO	DMAMUX request multiplexer channel 7 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH8_SO	DMAMUX request multiplexer channel 8 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH9_SO	DMAMUX request multiplexer channel 9 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH10_SO	DMAMUX request multiplexer channel 10 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_MUXCH11_SO	DMAMUX request multiplexer channel 11 synchronization overrun interrupt flag
DMAMUX_INT_FLAG_GENCH0_TO	DMAMUX request generator channel 0 trigger overrun interrupt flag
DMAMUX_INT_FLAG_GENCH1_TO	DMAMUX request generator channel 1 trigger overrun interrupt flag
DMAMUX_INT_FLAG_GENCH2_TO	DMAMUX request generator channel 2 trigger overrun interrupt flag

G_GENCH2_TO	
DMAMUX_INT_FLG	DMAMUX request generator channel 3 trigger overrun interrupt flag
G_GENCH3_TO	

## dma\_deinit

The description of dma\_deinit is shown as below:

**Table 3-236. Function dma\_deinit**

<b>Function name</b>	dma_deinit
<b>Function prototype</b>	void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	deinitialize DMA a channel registers
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<i>DMA_CHx</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## dma\_struct\_para\_init

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

**Table 3-237. Function dma\_struct\_para\_init**

<b>Function name</b>	dma_struct_para_init
<b>Function prototype</b>	void dma_struct_para_init(dma_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize the parameters of DMA struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>*init_struct</b>	A dma_parameter_struct address, refer to <a href="#">Table 3-227. Structure dma_parameter_struct</a>
<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 3-238. 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</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<i>DMA_CHx</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>init_struct</b>	A dma_parameter_struct address, the structure members refer to <a href="#">Table 3-227. Structure dma_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_32BIT;
dma_init_struct.number = TRANSFER_NUM/4;
dma_init_struct.periph_addr = (uint32_t)FLASH_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
```

```

dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_32BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA0, DMA_CH0, &dma_init_struct);

```

### dma\_circulation\_enable

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

**Table 3-239. Function dma\_circulation\_enable**

<b>Function name</b>	dma_circulation_enable
<b>Function prototype</b>	void dma_circulation_enable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	enable DMA circulation mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

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

```

### dma\_circulation\_disable

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

**Table 3-240. Function dma\_circulation\_disable**

<b>Function name</b>	dma_circulation_disable
<b>Function prototype</b>	void dma_circulation_disable(uint32_t dma_periph, dma_channel_enum channelx);
<b>Function descriptions</b>	disable DMA circulation mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx</i>	DMA peripheral selection
<b>Input parameter{in}</b>	

<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

**Table 3-241. 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</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<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 3-242. 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</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<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 3-243. 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</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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



## dma\_channel\_disable

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

**Table 3-244. 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</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-245. Function dma\_periph\_address\_config**

<b>Function name</b>	dma_periph_address_config
<b>Function prototype</b>	void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
<b>Function descriptions</b>	set DMA peripheral base address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>address</b>	peripheral base address

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

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

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

**Table 3-246. Function dma\_memory\_address\_config**

Function name	dma_memory_address_config
Function prototype	void dma_memory_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
Function descriptions	set DMA memory base address
Precondition	-
The called functions	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
Input parameter{in}	
<b>address</b>	memory base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
uint8_t g_destbuf[TRANSFER_NUM];

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

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

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

**Table 3-247. Function dma\_transfer\_number\_config**

Function name	dma_transfer_number_config
Function prototype	void dma_transfer_number_config(uint32_t dma_periph,

	<code>dma_channel_enum channelx, uint32_t number);</code>
<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</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>number</b>	data transfer number (0x00000000 – 0x0000FFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
#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 3-248. Function dma\_transfer\_number\_get**

<b>Function name</b>	<code>dma_transfer_number_get</code>
<b>Function prototype</b>	<code>uint32_t dma_transfer_number_get(uint32_t dma_periph, dma_channel_enum channelx);</code>
<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</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	0x00000000 – 0x0000FFFF

Example:

```
uint32_t number = 0;
```

```
number = dma_transfer_number_get(DMA0, DMA_CH0);
```

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

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

**Table 3-249. 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</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>priority</b>	priority Level of this channel
<i>DMA_PRIORITY_LOW</i>	low priority
<i>DMA_PRIORITY_MEDIUM</i>	medium priority
<i>DMA_PRIORITY_HIGH</i>	high priority
<i>DMA_PRIORITY_ULTRA_HIGH</i>	ultra high priority
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
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 3-250. 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</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>mwidth</b>	transfer data width of memory
<i>DMA_MEMORY_WIDT H_8BIT</i>	transfer data width of memory is 8-bit
<i>DMA_MEMORY_WIDT H_16BIT</i>	transfer data width of memory is 16-bit
<i>DMA_MEMORY_WIDT H_32BIT</i>	transfer data width of memory is 32-bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
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 3-251. 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>DMAx</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>pwidth</b>	transfer data width of peripheral
<i>DMA_PERIPHERAL_W</i>	transfer data width of peripheral is 8-bit

<i>IDTH_8BIT</i>	
<i>DMA_PERIPHERAL_W</i> <i>IDTH_16BIT</i>	transfer data width of peripheral is 16-bit
<i>DMA_PERIPHERAL_W</i> <i>IDTH_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 3-252. 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>DMAx</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<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 3-253. 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,

	<code>dma_channel_enum channelx</code> ;
<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><code>dma_periph</code></b>	DMA peripheral
<i>DMAx</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b><code>channelx</code></b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### dma\_periph\_increase\_enable

The description of `dma_periph_increase_enable` is shown as below:

**Table 3-254. Function `dma_periph_increase_enable`**

<b>Function name</b>	<code>dma_periph_increase_enable</code>
<b>Function prototype</b>	<code>void dma_periph_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);</code>
<b>Function descriptions</b>	enable next address increasement algorithm of peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b><code>dma_periph</code></b>	DMA peripheral
<i>DMAx</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b><code>channelx</code></b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## dma\_periph\_increase\_disable

The description of dma\_periph\_increase\_disable is shown as below:

**Table 3-255. 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</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<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 3-256. 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</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<b>Input parameter{in}</b>	
<b>direction</b>	specify the direction of data transfer
<i>DMA_PERIPHERAL_T</i>	read from peripheral and write to memory



<i>O_MEMORY</i>	
<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 3-257. 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</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<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 3-258. 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</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<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\_enable

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

**Table 3-259. 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</i>	DMA peripheral selection

Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### dma\_interrupt\_disable

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

**Table 3-260. 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>	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA0 channel0 interrupt configuration */

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

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

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

**Table 3-261. 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</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<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_ERR</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 3-262. 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</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx</i>	DMA channel selection, refer to <a href="#">Table 3-230. Enum dma_channel_enum</a>
<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);
}
```

### dmamux\_sync\_struct\_para\_init

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

**Table 3-263. Function dmamux\_sync\_struct\_para\_init**

<b>Function name</b>	dmamux_sync_struct_para_init
<b>Function prototype</b>	void dmamux_sync_struct_para_init(dmamux_sync_parameter_struct *init_struct);
<b>Function descriptions</b>	initialize the parameters of DMAMUX synchronization mode structure with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>init_struct</b>	the initialization data needed to initialize DMAMUX request multiplexer channel synchronization mode, refer to <a href="#">Table 3-228. Structure dmamux_sync_parameter_struct</a>
<b>Return value</b>	

-	-
---	---

Example:

```
/* initialize DMAMUX synchronization mode structure */

dmamux_sync_parameter_struct dmamux_sync_init_struct;

dmamux_sync_struct_para_init(&dmamux_sync_init_struct);
```

### dmamux\_synchronization\_init

The description of dmamux\_synchronization\_init is shown as below:

**Table 3-264. Function dmamux\_synchronization\_init**

Function name	dmamux_synchronization_init
Function prototype	void dmamux_synchronization_init(dmamux_multiplexer_channel_enum channelx, dmamux_sync_parameter_struct *init_struct);
Function descriptions	initialize DMAMUX request multiplexer channel synchronization mode
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMAMUX request multiplexer channel, refer to <a href="#">Table 3-231. Enum dmamux_multiplexer_channel_enum</a> .
Input parameter{in}	
init_struct	the initialization data needed to initialize DMAMUX request multiplexer channel synchronization mode, refer to <a href="#">Table 3-228. Structure dmamux_sync_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize DMAMUX synchronization mode structure */

dmamux_sync_parameter_struct dmamux_sync_init_struct;

dmamux_sync_struct_para_init(&dmamux_sync_init_struct);

/* initialize DMA request multiplexer channel 0 with synchronization mode */

dmamux_sync_init_struct.sync_id      = DMAMUX_SYNC_EXTI0;
dmamux_sync_init_struct.sync_polarity = DMAMUX_SYNC_RISING;
dmamux_sync_init_struct.request_number = 4;

dmamux_synchronization_init(DMAMUX_MUXCH0, &dmamux_sync_init_struct);
```

## dmamux\_synchronization\_enable

The description of dmamux\_synchronization\_enable is shown as below:

**Table 3-265. Function dmamux\_synchronization\_enable**

<b>Function name</b>	dmamux_synchronization_enable
<b>Function prototype</b>	void dmamux_synchronization_enable(dmamux_multiplexer_channel_enum channelx);
<b>Function descriptions</b>	enable synchronization mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMAMUX request multiplexer channel, refer to <a href="#">Table 3-231. Enum dmamux_multiplexer_channel_enum</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable synchronization mode */
```

```
dmamux_synchronization_enable(DMAMUX_MUXCH0);
```

## dmamux\_synchronization\_disable

The description of dmamux\_synchronization\_disable is shown as below:

**Table 3-266. Function dmamux\_synchronization\_disable**

<b>Function name</b>	dmamux_synchronization_disable
<b>Function prototype</b>	void dmamux_synchronization_disable(dmamux_multiplexer_channel_enum channelx);
<b>Function descriptions</b>	disable synchronization mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMAMUX request multiplexer channel, refer to <a href="#">Table 3-231. Enum dmamux_multiplexer_channel_enum</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable synchronization mode */
```

```
dmamux_synchronization_disable(DMAMUX_MUXCH0);
```

### dmamux\_event\_generation\_enable

The description of dmamux\_event\_generation\_enable is shown as below:

**Table 3-267. Function dmamux\_event\_generation\_enable**

<b>Function name</b>	dmamux_event_generation_enable
<b>Function prototype</b>	void dmamux_event_generation_enable(dmamux_multiplexer_channel_enum channelx);
<b>Function descriptions</b>	enable event generation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMAMUX request multiplexer channel, refer to <a href="#">Table 3-231. Enum dmamux_multiplexer_channel_enum</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable event generation */
```

```
dmamux_event_generation_enable(DMAMUX_MUXCH0);
```

### dmamux\_event\_generation\_disable

The description of dmamux\_event\_generation\_disable is shown as below:

**Table 3-268. Function dmamux\_event\_generation\_disable**

<b>Function name</b>	dmamux_event_generation_disable
<b>Function prototype</b>	void dmamux_event_generation_disable(dmamux_multiplexer_channel_enum channelx);
<b>Function descriptions</b>	disable event generation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMAMUX request multiplexer channel, refer to <a href="#">Table 3-231. Enum dmamux_multiplexer_channel_enum</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-



Example:

```
/* disable event generation */
dmamux_event_generation_disable(DMAMUX_MUXCH0);
```

### dmamux\_gen\_struct\_para\_init

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

**Table 3-269. Function dmamux\_gen\_struct\_para\_init**

Function name	dmamux_gen_struct_para_init
Function prototype	void dmamux_gen_struct_para_init(dmamux_gen_parameter_struct *init_struct);
Function descriptions	initialize the parameters of DMAMUX request generator structure with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_struct	the initialization data needed to initialize DMAMUX request generator channel, refer to <a href="#">Table 3-229. Structure dmamux_gen_parameter_struct</a> .
Return value	
-	-

Example:

```
/* initialize DMA request generator structure */
dmamux_gen_parameter_struct dmamux_gen_init_struct;
dmamux_gen_struct_para_init(&dmamux_gen_init_struct);
```

### dmamux\_request\_generator\_init

The description of dmamux\_request\_generator\_init is shown as below:

**Table 3-270. Function dmamux\_request\_generator\_init**

Function name	dmamux_request_generator_init
Function prototype	void dmamux_request_generator_init(dmamux_generator_channel_enum channelx, dmamux_gen_parameter_struct *init_struct);
Function descriptions	initialize DMAMUX request generator channel
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMAMUX generation channel, refer to <a href="#">Table 3-232. Enum dmamux_generator_channel_enum</a> .

Input parameter{in}	
<b>init_struct</b>	the initialization data needed to initialize DMAMUX request generator channel, refer to <a href="#">Table 3-229. Structure dmamux_gen_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize DMA request generator channel 0 */

dmamux_gen_parameter_struct    dmamux_gen_init_struct;

dmamux_gen_struct_para_init(&dmamux_gen_init_struct);

dmamux_gen_init_struct.trigger_id      = DMAMUX_TRIGGER_EXTI13;

dmamux_gen_init_struct.trigger_polarity = DMAMUX_GEN_RISING;

dmamux_gen_init_struct.request_number = 1;

dmamux_request_generator_init(DMAMUX_GENCH0, &dmamux_gen_init_struct);

```

### dmamux\_request\_generator\_channel\_enable

The description of dmamux\_request\_generator\_channel\_enable is shown as below:

**Table 3-271. Function dmamux\_request\_generator\_channel\_enable**

<b>Function name</b>	dmamux_request_generator_channel_enable
<b>Function prototype</b>	void dmamux_request_generator_channel_enable(dmamux_generator_channel_enum channelx);
<b>Function descriptions</b>	enable DMAMUX request generator channel
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>channelx</b>	DMAMUX generation channel, refer to <a href="#">Table 3-232. Enum dmamux_generator_channel_enum</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable DMAMUX request generator channel 0 */

dmamux_request_generator_channel_enable(DMAMUX_GENCH0);

```

## dmamux\_request\_generator\_channel\_disable

The description of dmamux\_request\_generator\_channel\_disable is shown as below:

**Table 3-272. Function dmamux\_request\_generator\_channel\_disable**

<b>Function name</b>	dmamux_request_generator_channel_disable
<b>Function prototype</b>	void dmamux_request_generator_channel_disable(dmamux_generator_channel_enum channelx);
<b>Function descriptions</b>	disable DMAMUX request generator channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMAMUX generation channel, refer to <a href="#">Table 3-232. Enum dmamux_generator_channel_enum</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMAMUX request generator channel 0 */
dmamux_request_generator_channel_disable(DMAMUX_GENCH0);
```

## dmamux\_synchronization\_polarity\_config

The description of dmamux\_synchronization\_polarity\_config is shown as below:

**Table 3-273. Function dmamux\_synchronization\_polarity\_config**

<b>Function name</b>	dmamux_synchronization_polarity_config
<b>Function prototype</b>	void dmamux_synchronization_polarity_config(dmamux_multiplexer_channel_enum channelx, uint32_t polarity);
<b>Function descriptions</b>	configure synchronization input polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMAMUX request multiplexer channel, refer to <a href="#">Table 3-231. Enum dmamux_multiplexer_channel_enum</a> .
<b>Input parameter{in}</b>	
<b>polarity</b>	synchronization input polarity
DMAMUX_SYNC_NO_EVENT	no event detection
DMAMUX_SYNC_RISING	rising edge

DMAMUX_SYNC_FALLING	falling edge
DMAMUX_SYNC_RISING_FALLING	rising and falling edges
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure synchronization input polarity */
```

```
dmamux_synchronization_polarity_config(DMAMUX_MUXCH0, DMAMUX_SYNC_RISING);
```

### dmamux\_request\_forward\_number\_config

The description of dmamux\_request\_forward\_number\_config is shown as below:

**Table 3-274. Function dmamux\_request\_forward\_number\_config**

Function name	dmamux_request_forward_number_config
Function prototype	void dmamux_request_forward_number_config(dmamux_multiplexer_channel_enum channelx, uint32_t number);
Function descriptions	configure number of DMA requests to forward
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMAMUX request multiplexer channel, refer to <a href="#">Table 3-231. Enum dmamux_multiplexer_channel_enum</a> .
Input parameter{in}	
number	DMA requests number to forward (1 - 32)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure number of DMA requests to forward */
```

```
dmamux_request_forward_number_config(DMAMUX_MUXCH0, 4);
```

### dmamux\_sync\_id\_config

The description of dmamux\_sync\_id\_config is shown as below:

Table 3-275. Function dmamux\_sync\_id\_config

Function name	dmamux_sync_id_config
Function prototype	void dmamux_sync_id_config(dmamux_multiplexer_channel_enum channelx, uint32_t id);
Function descriptions	configure synchronization input identification
Precondition	-
The called functions	-
Input parameter{in}	
channelx	DMAMUX request multiplexer channel, refer to <a href="#">Table 3-231. Enum dmamux_multiplexer_channel_enum</a> .
Input parameter{in}	
id	synchronization input identification
DMAMUX_SYNC_EXTI0	synchronization input is EXTI0
DMAMUX_SYNC_EXTI1	synchronization input is EXTI1
DMAMUX_SYNC_EXTI2	synchronization input is EXTI2
DMAMUX_SYNC_EXTI3	synchronization input is EXTI3
DMAMUX_SYNC_EXTI4	synchronization input is EXTI4
DMAMUX_SYNC_EXTI5	synchronization input is EXTI5
DMAMUX_SYNC_EXTI6	synchronization input is EXTI6
DMAMUX_SYNC_EXTI7	synchronization input is EXTI7
DMAMUX_SYNC_EXTI8	synchronization input is EXTI8
DMAMUX_SYNC_EXTI9	synchronization input is EXTI9
DMAMUX_SYNC_EXTI10	synchronization input is EXTI10
DMAMUX_SYNC_EXTI11	synchronization input is EXTI11
DMAMUX_SYNC_EXTI12	synchronization input is EXTI12
DMAMUX_SYNC_EXTI13	synchronization input is EXTI13
DMAMUX_SYNC_EXTI14	synchronization input is EXTI14
DMAMUX_SYNC_EXTI15	synchronization input is EXTI15

15	
DMAMUX_SYNC_EVT x_OUT0	synchronization input is Evt_out0
DMAMUX_SYNC_EVT x_OUT1	synchronization input is Evt_out1
DMAMUX_SYNC_EVT x_OUT2	synchronization input is Evt_out2
DMAMUX_SYNC_EVT x_OUT3	synchronization input is Evt_out3
DMAMUX_SYNC_LPTI MER_OUT	synchronization input is LPTIMER_OUT
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure synchronization input identification */
```

```
dmamux_sync_id_config(DMAMUX_MUXCH0, DMAMUX_SYNC_EXTI0);
```

### dmamux\_request\_id\_config

The description of dmamux\_request\_id\_config is shown as below:

**Table 3-276. Function dmamux\_request\_id\_config**

<b>Function name</b>	dmamux_request_id_config
<b>Function prototype</b>	void dmamux_request_id_config(dmamux_multiplexer_channel_enum channelx, uint32_t id);
<b>Function descriptions</b>	configure multiplexer input identification
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>channelx</b>	DMAMUX request multiplexer channel, refer to <a href="#">Table 3-231. Enum dmamux_multiplexer_channel_enum</a> .
Input parameter{in}	
<b>id</b>	input DMA request identification
DMA_REQUEST_M2M	memory to memory transfer
DMA_REQUEST_GENERATOR0	DMAMUX request generator 0
DMA_REQUEST_GENERATOR1	DMAMUX request generator 1
DMA_REQUEST_GENERATOR2	DMAMUX request generator 2

DMA_REQUEST_GENERATOR3	DMAMUX request generator 3
DMA_REQUEST_ADC0	DMAMUX request ADC0
DMA_REQUEST_DAC0_CH0	DMAMUX request DAC0_CH0
DMA_REQUEST_DAC0_CH1	DMAMUX request DAC0_CH1
DMA_REQUEST_TIMER5_UP	request DMAMUX TIMER5_UP
DMA_REQUEST_TIMER6_UP	request DMAMUX TIMER6_UP
DMA_REQUEST_SPI0_RX	request DMAMUX SPI0_RX
DMA_REQUEST_SPI0_TX	request DMAMUX SPI0_TX
DMA_REQUEST_SPI1_RX	request DMAMUX SPI1_RX
DMA_REQUEST_SPI1_TX	request DMAMUX SPI1_TX
DMA_REQUEST_SPI2_RX	request DMAMUX SPI2_RX
DMA_REQUEST_SPI2_TX	request DMAMUX SPI2_TX
DMA_REQUEST_I2C0_RX	request DMAMUX I2C0_RX
DMA_REQUEST_I2C0_TX	request DMAMUX I2C0_TX
DMA_REQUEST_I2C1_RX	request DMAMUX I2C1_RX
DMA_REQUEST_I2C1_TX	request DMAMUX I2C1_TX
DMA_REQUEST_I2C2_RX	request DMAMUX I2C2_RX
DMA_REQUEST_I2C2_TX	request DMAMUX I2C2_TX
DMA_REQUEST_I2C3_RX	request DMAMUX I2C3_RX
DMA_REQUEST_I2C3_TX	request DMAMUX I2C3_TX
DMA_REQUEST_USART0_RX	request DMAMUX USART0_RX
DMA_REQUEST_USART0_TX	request DMAMUX USART0_TX

ART0_TX	
DMA_REQUEST_US ART1_RX	request DMAMUX USART1_RX
DMA_REQUEST_US ART1_TX	request DMAMUX USART1_TX
DMA_REQUEST_US ART2_RX	request DMAMUX USART2_RX
DMA_REQUEST_US ART2_TX	request DMAMUX USART2_TX
DMA_REQUEST_UA RT3_RX	request DMAMUX UART3_RX
DMA_REQUEST_UA RT3_TX	request DMAMUX UART3_TX
DMA_REQUEST_UA RT4_RX	request DMAMUX UART4_RX
DMA_REQUEST_UA RT4_TX	request DMAMUX UART4_TX
DMA_REQUEST_AD C1	request DMAMUX ADC1
DMA_REQUEST_AD C2	request DMAMUX ADC2
DMA_REQUEST_AD C3	request DMAMUX ADC3
DMA_REQUEST_QS PI	request DMAMUX QSPI
DMA_REQUEST_DA C1_CH0	request DMAMUX DAC1_CH0
DMA_REQUEST_DA C1_CH1	request DMAMUX DAC1_CH1
DMA_REQUEST_TI MER0_CH0	request DMAMUX TIMER0_CH0
DMA_REQUEST_TI MER0_CH1	request DMAMUX TIMER0_CH1
DMA_REQUEST_TI MER0_CH2	request DMAMUX TIMER0_CH2
DMA_REQUEST_TI MER0_CH3	request DMAMUX TIMER0_CH3
DMA_REQUEST_TI MER0_CH0N	request DMAMUX TIMER0_CH0N
DMA_REQUEST_TI MER0_CH1N	request DMAMUX TIMER0_CH1N
DMA_REQUEST_TI MER0_CH2N	request DMAMUX TIMER0_CH2N



DMA_REQUEST_TIMER0_CH3N	request DMAMUX TIMER0_CH3N
DMA_REQUEST_TIMER0_UP	request DMAMUX TIMER0_UP
DMA_REQUEST_TIMER0_TRIG	request DMAMUX TIMER0_TRIG
DMA_REQUEST_TIMER0_COM	request DMAMUX TIMER0_COM
DMA_REQUEST_TIMER7_CH0	request DMAMUX TIMER7_CH0
DMA_REQUEST_TIMER7_CH1	request DMAMUX TIMER7_CH1
DMA_REQUEST_TIMER7_CH2	request DMAMUX TIMER7_CH2
DMA_REQUEST_TIMER7_CH3	request DMAMUX TIMER7_CH3
DMA_REQUEST_TIMER7_CH0N	request DMAMUX TIMER7_CH0N
DMA_REQUEST_TIMER7_CH1N	request DMAMUX TIMER7_CH1N
DMA_REQUEST_TIMER7_CH2N	request DMAMUX TIMER7_CH2N
DMA_REQUEST_TIMER7_CH3N	request DMAMUX TIMER7_CH3N
DMA_REQUEST_TIMER7_UP	request DMAMUX TIMER7_UP
DMA_REQUEST_TIMER7_TRIG	request DMAMUX TIMER7_TRIG
DMA_REQUEST_TIMER7_COM	request DMAMUX TIMER7_COM
DMA_REQUEST_TIMER1_CH0	request DMAMUX TIMER1_CH0
DMA_REQUEST_TIMER1_CH1	request DMAMUX TIMER1_CH1
DMA_REQUEST_TIMER1_CH2	request DMAMUX TIMER1_CH2
DMA_REQUEST_TIMER1_CH3	request DMAMUX TIMER1_CH3
DMA_REQUEST_TIMER1_UP	request DMAMUX TIMER1_UP
DMA_REQUEST_TIMER1_TRIG	request DMAMUX TIMER1_TRIG
DMA_REQUEST_TIMER2_CH0	request DMAMUX TIMER2_CH0

MER2_CH0	
DMA_REQUEST_TI MER2_CH1	request DMAMUX TIMER2_CH1
DMA_REQUEST_TI MER2_CH2	request DMAMUX TIMER2_CH2
DMA_REQUEST_TI MER2_CH3	request DMAMUX TIMER2_CH3
DMA_REQUEST_TI MER2_UP	request DMAMUX TIMER2_UP
DMA_REQUEST_TI MER2_TRIG	request DMAMUX TIMER2_TRIG
DMA_REQUEST_TI MER3_CH0	request DMAMUX TIMER3_CH0
DMA_REQUEST_TI MER3_CH1	request DMAMUX TIMER3_CH1
DMA_REQUEST_TI MER3_CH2	request DMAMUX TIMER3_CH2
DMA_REQUEST_TI MER3_CH3	request DMAMUX TIMER3_CH3
DMA_REQUEST_TI MER3_UP	request DMAMUX TIMER3_UP
DMA_REQUEST_TI MER3_TRIG	request DMAMUX TIMER3_TRIG
DMA_REQUEST_TI MER4_CH0	request DMAMUX TIMER4_CH0
DMA_REQUEST_TI MER4_CH1	request DMAMUX TIMER4_CH1
DMA_REQUEST_TI MER4_CH2	request DMAMUX TIMER4_CH2
DMA_REQUEST_TI MER4_CH3	request DMAMUX TIMER4_CH3
DMA_REQUEST_TI MER4_UP	request DMAMUX TIMER4_UP
DMA_REQUEST_TI MER4_TRIG	request DMAMUX TIMER4_TRIG
DMA_REQUEST_TI MER14_CH0	request DMAMUX TIMER14_CH0
DMA_REQUEST_TI MER14_CH1	request DMAMUX DMAMUXTIMER14_CH1
DMA_REQUEST_TI MER14_CH0N	request DMAMUX TIMER14_CH0N
DMA_REQUEST_TI MER14_UP	request DMAMUX TIMER14_UP

DMA_REQUEST_TIMER14_TRIG	request DMAMUX TIMER14_TRIG
DMA_REQUEST_TIMER14_COM	request DMAMUX TIMER14_COM
DMA_REQUEST_TIMER15_CH0	request DMAMUX TIMER15_CH0
DMA_REQUEST_TIMER15_CH0N	request DMAMUX TIMER15_CH0N
DMA_REQUEST_TIMER15_UP	request DMAMUX TIMER15_UP
DMA_REQUEST_TIMER16_CH0	request DMAMUX TIMER16_CH0
DMA_REQUEST_TIMER16_CH0N	request DMAMUX TIMER16_CH0N
DMA_REQUEST_TIMER16_UP	request DMAMUX TIMER16_UP
DMA_REQUEST_TIMER19_CH0	request DMAMUX TIMER19_CH0
DMA_REQUEST_TIMER19_CH1	request DMAMUX TIMER19_CH1
DMA_REQUEST_TIMER19_CH2	request DMAMUX TIMER19_CH2
DMA_REQUEST_TIMER19_CH3	request DMAMUX TIMER19_CH3
DMA_REQUEST_TIMER19_CH0N	request DMAMUX TIMER19_CH0N
DMA_REQUEST_TIMER19_CH1N	request DMAMUX TIMER19_CH1N
DMA_REQUEST_TIMER19_CH2N	request DMAMUX TIMER19_CH2N
DMA_REQUEST_TIMER19_CH3N	request DMAMUX TIMER19_CH3N
DMA_REQUEST_TIMER19_UP	request DMAMUX TIMER19_UP
DMA_REQUEST_TIMER19_TRIG	request DMAMUX TIMER19_TRIG
DMA_REQUEST_TIMER19_COM	request DMAMUX TIMER19_COM
DMA_REQUEST_CAU_IN	request DMAMUX CAU_IN
DMA_REQUEST_CAU_OUT	request DMAMUX CAU_OUT
DMA_REQUEST_HRTIMER_MASTER	request DMAMUX HRTIMER_MASTER

TIMER_MASTER	
DMA_REQUEST_HRTIMER_TIMER0	request DMAMUX HRTIMER_TIMER0
DMA_REQUEST_HRTIMER_TIMER1	request DMAMUX HRTIMER_TIMER1
DMA_REQUEST_HRTIMER_TIMER2	request DMAMUX HRTIMER_TIMER2
DMA_REQUEST_HRTIMER_TIMER3	request DMAMUX HRTIMER_TIMER3
DMA_REQUEST_HRTIMER_TIMER4	request DMAMUX HRTIMER_TIMER4
DMA_REQUEST_HRTIMER_TIMER5	request DMAMUX HRTIMER_TIMER5
DMA_REQUEST_HRTIMER_TIMER6	request DMAMUX HRTIMER_TIMER6
DMA_REQUEST_HRTIMER_TIMER7	request DMAMUX HRTIMER_TIMER7
DMA_REQUEST_DAC2_CH0	request DMAMUX DAC2_CH0
DMA_REQUEST_DAC2_CH1	request DMAMUX DAC2_CH1
DMA_REQUEST_DAC3_CH0	request DMAMUX DAC3_CH0
DMA_REQUEST_DAC3_CH1	request DMAMUX DAC3_CH1
DMA_REQUEST_HP_DF_FLT0	request DMAMUX HPDF_FLT0
DMA_REQUEST_HP_DF_FLT1	request DMAMUX HPDF_FLT1
DMA_REQUEST_HP_DF_FLT2	request DMAMUX HPDF_FLT2
DMA_REQUEST_HP_DF_FLT3	request DMAMUX HPDF_FLT3
DMA_REQUEST_FAC_READ	request DMAMUX FAC_READ
DMA_REQUEST_FAC_WRITE	request DMAMUX FAC_WRITE
DMA_REQUEST_TMU_INPUT	request DMAMUX TMU_INPUT
DMA_REQUEST_TMU_OUTPUT	request DMAMUX TMU_OUTPUT
DMA_REQUEST_CAN0	request DMAMUX CAN0

DMA_REQUEST_CAN1	request DMAMUX CAN1
DMA_REQUEST_CAN2	request DMAMUX CAN2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure multiplexer input identification */
```

```
dmamux_request_id_config(DMAMUX_MUXCH0, DMA_REQUEST_GENERATOR0);
```

### dmamux\_trigger\_polarity\_config

The description of dmamux\_trigger\_polarity\_config is shown as below:

**Table 3-277. Function dmamux\_trigger\_polarity\_config**

<b>Function name</b>	dmamux_trigger_polarity_config
<b>Function prototype</b>	void dmamux_trigger_polarity_config(dmamux_generator_channel_enum channelx, uint32_t polarity);
<b>Function descriptions</b>	configure trigger input polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>channelx</b>	DMAMUX generation channel, refer to <a href="#">Table 3-232. Enum dmamux_generator_channel_enum</a> .
Input parameter{in}	
<b>polarity</b>	trigger input polarity
DMAMUX_GEN_NO_EVENT	no event detection
DMAMUX_GEN_RISING	rising edge
DMAMUX_GEN_FALLING	falling edge
DMAMUX_GEN_RISING_FALLING	rising and falling edges
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure trigger input polarity */
```

```
dmamux_trigger_polarity_config(DMAMUX_GENCH0, DMAMUX_GEN_RISING);
```

### dmamux\_request\_generate\_number\_config

The description of dmamux\_request\_generate\_number\_config is shown as below:

**Table 3-278. Function dmamux\_request\_generate\_number\_config**

<b>Function name</b>	dmamux_request_generate_number_config
<b>Function prototype</b>	void dmamux_request_generate_number_config(dmamux_generator_channel_enum channelx, uint32_t number);
<b>Function descriptions</b>	configure number of DMA requests to be generated
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMAMUX generation channel, refer to <a href="#">Table 3-232. Enum dmamux_generator_channel_enum</a> .
<b>Input parameter{in}</b>	
<b>number</b>	DMA requests number to be generated (1 - 32)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure number of DMA requests to be generated */
```

```
dmamux_request_generate_number_config(DMAMUX_GENCH0, 1);
```

### dmamux\_trigger\_id\_config

The description of dmamux\_trigger\_id\_config is shown as below:

**Table 3-279. Function dmamux\_trigger\_id\_config**

<b>Function name</b>	dmamux_trigger_id_config
<b>Function prototype</b>	void dmamux_trigger_id_config(dmamux_generator_channel_enum channelx, uint32_t id);
<b>Function descriptions</b>	configure trigger input identification
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMAMUX generation channel, refer to <a href="#">Table 3-232. Enum dmamux_generator_channel_enum</a> .
<b>Input parameter{in}</b>	
<b>id</b>	trigger input identification
<i>DMAMUX_TRIGGER_</i>	trigger input is EXTI0

<i>EXTI0</i>	
<i>DMAMUX_TRIGGER_</i> <i>EXTI1</i>	trigger input is EXTI1
<i>DMAMUX_TRIGGER_</i> <i>EXTI2</i>	trigger input is EXTI2
<i>DMAMUX_TRIGGER_</i> <i>EXTI3</i>	trigger input is EXTI3
<i>DMAMUX_TRIGGER_</i> <i>EXTI4</i>	trigger input is EXTI4
<i>DMAMUX_TRIGGER_</i> <i>EXTI5</i>	trigger input is EXTI5
<i>DMAMUX_TRIGGER_</i> <i>EXTI6</i>	trigger input is EXTI6
<i>DMAMUX_TRIGGER_</i> <i>EXTI7</i>	trigger input is EXTI7
<i>DMAMUX_TRIGGER_</i> <i>EXTI8</i>	trigger input is EXTI8
<i>DMAMUX_TRIGGER_</i> <i>EXTI9</i>	trigger input is EXTI9
<i>DMAMUX_TRIGGER_</i> <i>EXTI10</i>	trigger input is EXTI10
<i>DMAMUX_TRIGGER_</i> <i>EXTI11</i>	trigger input is EXTI11
<i>DMAMUX_TRIGGER_</i> <i>EXTI12</i>	trigger input is EXTI12
<i>DMAMUX_TRIGGER_</i> <i>EXTI13</i>	trigger input is EXTI13
<i>DMAMUX_TRIGGER_</i> <i>EXTI14</i>	trigger input is EXTI14
<i>DMAMUX_TRIGGER_</i> <i>EXTI15</i>	trigger input is EXTI15
<i>DMAMUX_TRIGGER_</i> <i>EVT_OUT0</i>	trigger input is Evt_out0
<i>DMAMUX_TRIGGER_</i> <i>EVT_OUT1</i>	trigger input is Evt_out1
<i>DMAMUX_TRIGGER_</i> <i>EVT_OUT2</i>	trigger input is Evt_out2
<i>DMAMUX_TRIGGER_</i> <i>EVT_OUT3</i>	trigger input is Evt_out3
<i>DMAMUX_TRIGGER_</i> <i>EVT_OUT4</i>	trigger input is Evt_out4
<i>DMAMUX_TRIGGER_</i> <i>EVT_OUT5</i>	trigger input is Evt_out5

<i>DMAMUX_TRIGGER_EVT_OUT6</i>	trigger input is Evt_out6
<i>DMAMUX_TRIGGER_EVT_OUT7</i>	trigger input is Evt_out7
<i>DMAMUX_TRIGGER_RTC_WAKEUP</i>	trigger input is wakeup
<i>DMAMUX_TRIGGER_CMP0_OUTPUT</i>	trigger input is CMP0 output
<i>DMAMUX_TRIGGER_CMP1_OUTPUT</i>	trigger input is CMP1 output
<i>DMAMUX_TRIGGER_I2C0_WAKEUP</i>	trigger input is I2C0 wakeup
<i>DMAMUX_TRIGGER_I2C1_WAKEUP</i>	trigger input is I2C1 wakeup
<i>DMAMUX_TRIGGER_I2C2_WAKEUP</i>	trigger input is I2C2 wakeup
<i>DMAMUX_TRIGGER_I2C3_WAKEUP</i>	trigger input is I2C3 wakeup
<i>DMAMUX_TRIGGER_I2C0_INT_EVENT</i>	trigger input is I2C0 interrupt event
<i>DMAMUX_TRIGGER_I2C1_INT_EVENT</i>	trigger input is I2C1 interrupt event
<i>DMAMUX_TRIGGER_I2C2_INT_EVENT</i>	trigger input is I2C2 interrupt event
<i>DMAMUX_TRIGGER_I2C3_INT_EVENT</i>	trigger input is I2C3 interrupt event
<i>DMAMUX_TRIGGER_ADC2_INT</i>	ADC2 interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure trigger input identification */
```

```
dmamux_trigger_id_config(DMAMUX_GENCH0, DMAMUX_TRIGGER_EXTI13);
```

### dmamux\_flag\_get

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

**Table 3-280. Function dmamux\_flag\_get**

<b>Function name</b>	dmamux_flag_get
<b>Function prototype</b>	FlagStatus dmamux_flag_get(dmamux_flag_enum flag);



<b>Function descriptions</b>	get DMAMUX flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flag type, refer to <a href="#">Table 3-234. Enum dmamux_flag_enum.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get DMAMUX flag */
```

```
FlagStatus flag = RESET;
```

```
flag = dmamux_flag_get(DMAMUX_FLAG_GENCH0_TO);
```

### dmamux\_flag\_clear

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

**Table 3-281. Function dmamux\_flag\_clear**

<b>Function name</b>	dmamux_flag_clear
<b>Function prototype</b>	void dmamux_flag_clear(dmamux_flag_enum flag);
<b>Function descriptions</b>	clear DMAMUX flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flag type, refer to <a href="#">Table 3-234. Enum dmamux_flag_enum.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear DMAMUX flag */
```

```
dmamux_flag_clear(DMAMUX_FLAG_GENCH0_TO);
```

### dmamux\_interrupt\_enable

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

**Table 3-282. Function dmamux\_interrupt\_enable**

<b>Function name</b>	dmamux_interrupt_enable
<b>Function prototype</b>	void dmamux_interrupt_enable(dmamux_interrupt_enum interrupt);
<b>Function descriptions</b>	enable DMAMUX interrupt

Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify which interrupt to enable, refer to <a href="#">Table 3-233. Enum dmamux_interrupt_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMAMUX interrupt */
dmamux_interrupt_enable(DMAMUX_INT_MUXCH0_SO);
```

### dmamux\_interrupt\_disable

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

**Table 3-283. Function dmamux\_interrupt\_disable**

Function name	dmamux_interrupt_disable
Function prototype	void dmamux_interrupt_disable(dmamux_interrupt_enum interrupt);
Function descriptions	disable DMAMUX interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify which interrupt to disable, refer to <a href="#">Table 3-233. Enum dmamux_interrupt_enum</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMAMUX interrupt */
dmamux_interrupt_disable(DMAMUX_INT_MUXCH0_SO);
```

### dmamux\_interrupt\_flag\_get

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

**Table 3-284. Function dmamux\_interrupt\_flag\_get**

Function name	dmamux_interrupt_flag_get
Function prototype	FlagStatus dmamux_interrupt_flag_get(dmamux_interrupt_flag_enum int_flag);

<b>Function descriptions</b>	get DMAMUX interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	flag type, refer to <a href="#">Table 3-235. Enum dmamux_interrupt_flag_enum.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get DMAMUX interrupt flag */

FlagStatus flag = RESET;

flag = dmamux_interrupt_flag_get(DMAMUX_INT_FLAG_MUXCH0_SO);
```

### dmamux\_interrupt\_flag\_clear

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

**Table 3-285. Function dmamux\_interrupt\_flag\_clear**

<b>Function name</b>	dmamux_interrupt_flag_clear
<b>Function prototype</b>	FlagStatus dmamux_interrupt_flag_get(dmamux_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	clear DMAMUX interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	flag type, refer to <a href="#">Table 3-235. Enum dmamux_interrupt_flag_enum.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear DMAMUX interrupt flag */

dmamux_interrupt_flag_clear(DMAMUX_INT_FLAG_MUXCH0_SO);
```

## 3.12. EXMC

The external memory controller EXMC, is used as a translator for MCU to access a variety of external memory. The EXMC registers are listed in chapter [3.12.1](#), the EXMC firmware

functions are introduced in chapter [3.12.2](#).

### 3.12.1. Descriptions of Peripheral registers

EXMC registers are listed in the table shown as below:

**Table 3-286. EXMC Registers**

Registers	Descriptions
EXMC_PNCTL	PSRAM/NOR Flash control registers
EXMC_PNTCFG	PSRAM/NOR Flash timing configuration registers
EXMC_NCTL	NAND flash control registers
EXMC_NSTAT	NAND flash interrupt enable registers
EXMC_NCTCFG	NAND flash common space timing configuration registers
EXMC_NATCFG	NAND flash attribute space timing configuration registers
EXMC_NECC	NAND flash ECC registers

### 3.12.2. Descriptions of Peripheral functions

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

**Table 3-287. 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
exmc_nand_enable	enable EXMC NAND bankx
exmc_nand_disable	disable EXMC NAND bankx
exmc_pccard_deinit	deinitialize EXMC PC card bank

Function name	Function description
exmc_pccard_struct_para_init	initialize exmc_pccard_parameter_struct with the default values
exmc_norsram_page_size_config	configure CRAM page size
exmc_nand_ecc_enable	enable the EXMC NAND ECC function
exmc_nand_ecc_disable	disable the EXMC NAND ECC function
exmc_ecc_get	get the EXMC ECC value
exmc_flag_get	get EXMC flag status

### Structure exmc\_norsram\_timing\_parameter\_struct

**Table 3-288. Structure exmc\_norsram\_timing\_parameter\_struct**

Member name	Function description
syn_data_latency	configure the data latency
syn_clk_division	configure the clock divide ratio
bus_latency	configure the bus latency
asyn_data_setup_time	configure the data setup time,asynchronous access mode valid
asyn_address_hold_time	configure the address hold time,asynchronous access mode valid
asyn_address_setup_time	configure the data setup time,asynchronous access mode valid

### Structure exmc\_norsram\_parameter\_struct

**Table 3-289. Structure exmc\_norsram\_parameter\_struct**

Member name	Function description
write_mode	the write mode, synchronous mode or asynchronous mode
asyn_wait	enable or disable the asynchronous wait function
nwait_signal	enable or disable the NWAIT signal while in synchronous bust mode
memory_write	enable or disable the write operation
nwait_config	NWAIT signal configuration
wrap_burst_mode	enable or disable the wrap burst mode

Member name	Function description
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

### Structure exmc\_nand\_timing\_parameter\_struct

**Table 3-290. Structure exmc\_nand\_timing\_parameter\_struct**

Member name	Function description
databus_hiztime	configure the databus HiZ time for write operation
holdtime	configure the address hold time(or the data hold time for write operation)
waittime	configure the minimum wait time
setuptime	configure the address setup time

### Structure exmc\_nand\_parameter\_struct

**Table 3-291. 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

## exmc\_norsram\_deinit

The description of exmc\_norsram\_deinit is shown as below:

**Table 3-292. Function exmc\_norsram\_deinit**

<b>Function name</b>	exmc_norsram_deinit
<b>Function prototype</b>	void exmc_norsram_deinit(void);
<b>Function descriptions</b>	deinitialize EXMC NOR/SRAM region
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* deinitialize the EXMC NOR/SRAM region0 of bank0 */
```

```
exmc_norsram_deinit();
```

## exmc\_norsram\_struct\_para\_init

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

**Table 3-293. Function exmc\_norsram\_struct\_para\_init**

<b>Function name</b>	exmc_norsram_struct_para_init
<b>Function prototype</b>	void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
<b>Function descriptions</b>	initialize the struct exmc_norsram_parameter_struct
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_norsram_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-289. Structure exmc_norsram_parameter_struct</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 3-294. 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}	
<b>exmc_norsram_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-289. Structure exmc_norsram_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize EXMC NOR/PSRAM bank */
exmc_norsram_parameter_struct lcd_init_struct;
exmc_norsram_timing_parameter_struct lcd_timing_init_struct;
/* configure timing parameter */
```



```

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.write_mode = EXMC_ASYN_WRITE;

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_NOR;

lcd_init_struct.address_data_mux = DISABLE;

lcd_init_struct.read_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 3-295. Function exmc\_norsram\_enable**

<b>Function name</b>	exmc_norsram_enable
<b>Function prototype</b>	void exmc_norsram_enable(void);
<b>Function descriptions</b>	enable EXMC NOR/PSRAM region
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

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

```
exmc_norsram_enable();
```

### exmc\_norsram\_disable

The description of exmc\_norsram\_disable is shown as below:

**Table 3-296. Function exmc\_norsram\_disable**

<b>Function name</b>	exmc_norsram_disable
<b>Function prototype</b>	void exmc_norsram_disable(void);
<b>Function descriptions</b>	disable EXMC NOR/PSRAM region
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
exmc_norsram_disable();
```

### exmc\_nand\_deinit

The description of exmc\_nand\_deinit is shown as below:

Table 3-297. Function exmc\_nand\_deinit

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

Example:

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

```
exmc_norsram_deinit();
```

### exmc\_nand\_struct\_para\_init

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

Table 3-298. Function exmc\_nand\_struct\_para\_init

Function name	exmc_nand_struct_para_init
Function prototype	void exmc_nand_struct_para_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
Function descriptions	initialize the struct exmc_nand_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_init_struct	Structure for initialization, the structure members can refer to <a href="#">Table 3-291. Structure exmc_nand_parameter_struct</a>
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* initialize the struct nand_init_struct */
exmc_nand_parameter_struct nand_init_struct;
exmc_nand_struct_para_init (&nand_init_struct);
```

## exmc\_nand\_init

The description of exmc\_nand\_init is shown as below:

**Table 3-299. Function exmc\_nand\_init**

Function name	exmc_nand_init
Function prototype	void exmc_nand_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
Function descriptions	initialize EXMC NAND bank
Precondition	-
The called functions	-
Input parameter{in}	
<b>exmc_nand_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-291. Structure exmc_nand_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
exmc_nand_parameter_struct nand_init_struct;
exmc_nand_pccard_timing_parameter_struct nand_timing_init_struct;

/* EXMC configuration */
nand_timing_init_struct.setuptime = 5;
nand_timing_init_struct.waittime = 4;
nand_timing_init_struct.holdtime = 2;
```

```

nand_timing_init_struct.databus_hiztime = 2;

nand_init_struct.nand_bank = EXMC_BANK1_NAND;

nand_init_struct.ecc_size = EXMC_ECC_SIZE_2048BYTES;

nand_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;

nand_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;

nand_init_struct.ecc_logic = ENABLE;

nand_init_struct.databus_width = EXMC_NAND_DATABUS_WIDTH_8B;

nand_init_struct.wait_feature = ENABLE;

nand_init_struct.common_space_timing = &nand_timing_init_struct;

nand_init_struct.attribute_space_timing = &nand_timing_init_struct;

exmc_nand_init(&nand_init_struct);

```

### exmc\_nand\_enable

The description of exmc\_nand\_enable is shown as below:

**Table 3-300. Function exmc\_nand\_enable**

<b>Function name</b>	exmc_nand_enable
<b>Function prototype</b>	void exmc_nand_enable(void);
<b>Function descriptions</b>	enable EXMC NAND 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 NAND bank1 */

exmc_nand_enable();

```

## exmc\_nand\_disable

The description of exmc\_nand\_disable is shown as below:

**Table 3-301. Function exmc\_nand\_disable**

<b>Function name</b>	exmc_nand_disable
<b>Function prototype</b>	exmc_nand_disable(void);
<b>Function descriptions</b>	disable EXMC NAND 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 NAND bank1 */
exmc_nand_disable();
```

## exmc\_norsram\_page\_size\_config

The description of exmc\_norsram\_page\_size\_config is shown as below:

**Table 3-302. 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
<i>EXMC_CRAM_AUTO_SPLIT</i>	the clock is generated only during synchronous access

EXMC_CRAM_PAGE_SIZE_128_BYTES	page size is 128 bytes
EXMC_CRAM_PAGE_SIZE_256_BYTES	page size is 256 bytes
EXMC_CRAM_PAGE_SIZE_512_BYTES	page size is 512 bytes
EXMC_CRAM_PAGE_SIZE_1024_BYTES	page size is 1024 bytes
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CRAM page size */
```

```
exmc_norsram_page_size_config (EXMC_CRAM_PAGE_SIZE_128_BYTES);
```

### exmc\_nand\_ecc\_enable

The description of exmc\_nand\_ecc\_enable is shown as below:

**Table 3-303. Function exmc\_nand\_ecc\_enable**

Function name	exmc_nand_ecc_enable
Function prototype	void exmc_nand_ecc_enable(void);
Function descriptions	enable the EXMC NAND ECC function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
exmc_nand_ecc_enable();
```

### exmc\_nand\_ecc\_disable

The description of exmc\_nand\_ecc\_disable is shown as below:

**Table 3-304. Function exmc\_nand\_ecc\_disable**

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

Example:

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

```
exmc_nand_ecc_disable ();
```

### exmc\_ecc\_get

The description of exmc\_ecc\_get is shown as below:

**Table 3-305. Function exmc\_ecc\_get**

<b>Function name</b>	exmc_ecc_get
<b>Function prototype</b>	uint32_t exmc_ecc_get(uint32_t exmc_nand_bank);
<b>Function descriptions</b>	get the EXMC ECC value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	



-	-
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\_flag\_get

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

**Table 3-306. Function exmc\_flag\_get**

<b>Function name</b>	exmc_flag_get
<b>Function prototype</b>	FlagStatus exmc_flag_get(uint32_t flag);
<b>Function descriptions</b>	get EXMC flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>flag</b>	specify get which flag
EXMC_NAND_FLAG_FIFOE	FIFO empty status
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check FIFO status is set or not*/

if(RESET != exmc_flag_get (EXMC_NAND_PCCARD_FLAG_FIFOE));
```

## 3.13. EXTI

EXTI is the interrupt / event controller in the MCU. It contains up to 20 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.13.1](#), the EXTI firmware functions are introduced in chapter [3.13.2](#).

### 3.13.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

**Table 3-307. EXTI Registers**

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

### 3.13.2. Descriptions of Peripheral functions

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

**Table 3-308. EXTI firmware function**

Function name	Function description
exti_deinit	deinitialize the EXTI
exti_init	initialize the EXTI line x
exti_interrupt_enable	enable the interrupts from EXTI line x
exti_interrupt_disable	disable the interrupts from EXTI line x
exti_event_enable	enable the events from EXTI line x
exti_event_disable	disable the events from EXTI line x
exti_software_interrupt_enable	enable the software interrupt event from EXTI line x
exti_software_interrupt_disable	enable 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 3-309. Enum exti\_line\_enum**

Member name	Function description
EXTI_0	EXTI line 0
EXTI_1	EXTI line 1

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

### Enum exti\_mode\_enum

Table 3-310. Enum exti\_mode\_enum

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

### Enum exti\_trig\_type\_enum

Table 3-311. 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 3-312. 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 3-313. Function exti\_init**

Function name	exti_init
Function prototype	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
Function descriptions	initialize the EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-309. Enum exti_line_enum</a>
Input parameter{in}	
mode	EXTI mode, refer to <a href="#">Table 3-310. Enum exti_mode_enum</a>
Input parameter{in}	
trig_type	trigger type, refer to <a href="#">Table 3-311. 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 3-314. Function exti\_interrupt\_enable**

Function name	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 3-309. 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 3-315. 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 3-309. 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 3-316. 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

Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-309. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
exti_event_enable(EXTI_0);
```

### exti\_event\_disable

The description of exti\_event\_disable is shown as below:

**Table 3-317. Function exti\_event\_disable**

Function name	exti_event_disable
Function prototype	void exti_event_disable(exti_line_enum linex);
Function descriptions	disable the events from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-309. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
exti_event_disable(EXTI_0);
```

### exti\_software\_interrupt\_enable

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

**Table 3-318. Function exti\_software\_interrupt\_enable**

Function name	exti_software_interrupt_enable
Function prototype	void exti_software_interrupt_enable(exti_line_enum linex);
Function descriptions	enable the software interrupt event from EXTI line x
Precondition	-
The called functions	-

Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-309. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
exti_software_interrupt_enable(EXTI_0);
```

### exti\_software\_interrupt\_disable

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

**Table 3-319. Function exti\_software\_interrupt\_disable**

Function name	exti_software_interrupt_disable
Function prototype	void exti_software_interrupt_disable(exti_line_enum linex);
Function descriptions	disable the software interrupt event from EXTI line
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-309. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
exti_software_interrupt_disable(EXTI_0);
```

### exti\_flag\_get

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

**Table 3-320. Function exti\_flag\_get**

Function name	exti_flag_get
Function prototype	FlagStatus exti_flag_get(exti_line_enum linex);
Function descriptions	get EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-309. Enum exti_line_enum</a>

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

Example:

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

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

### exti\_flag\_clear

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

**Table 3-321. Function exti\_flag\_clear**

Function name	exti_flag_clear
Function prototype	void exti_flag_clear(exti_line_enum linex);
Function descriptions	clear EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-309. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
exti_flag_clear(EXTI_0);
```

### exti\_interrupt\_flag\_get

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

**Table 3-322. Function exti\_interrupt\_flag\_get**

Function name	exti_interrupt_flag_get
Function prototype	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
Function descriptions	get EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-309. Enum exti_line_enum</a>
Output parameter{out}	
-	-



Return value	
FlagStatus	SET or RESET

Example:

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

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

### exti\_interrupt\_flag\_clear

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

**Table 3-323. Function exti\_interrupt\_flag\_clear**

Function name	exti_interrupt_flag_clear
Function prototype	void exti_interrupt_flag_clear(exti_line_enum linex);
Function descriptions	clear EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-309. Enum exti_line_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
exti_interrupt_flag_clear(EXTI_0);
```

## 3.14. FMC

There is flash controller and option byte for GD32E50x series. The FMC registers are listed in chapter [3.14.1](#) the FMC firmware functions are introduced in chapter [3.14.2](#).

### 3.14.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

**Table 3-324. FMC Registers**

Registers	Descriptions
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_OBCTL0	FMC option byte control register 0
FMC_OBCTL1	FMC option byte control register 1
FMC_OBCTL2	FMC option byte control register 2
FMC_OTP1CFG	FMC OTP1 configuration register
FMC_OBSTAT	FMC option bytes status 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_OTP3_STAT	FMC OTP3 status register
FMC_PID	FMC product ID register
OB_SPC	option byte security protection value
OB_USER	option byte user value
OB_DATA1	option byte data1 value
OB_DATA2	option byte data2 value
OB_WP0	option byte write protection 0
OB_WP1	option byte write protection 1
OB_WP2	option byte write protection 2
OB_WP3	option byte write protection 3

### 3.14.2. Descriptions of Peripheral functions

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

**Table 3-325. FMC firmware function**

Function name	Function description
fmc_unlock	unlock the main FMC operation
fmc_bank0_unlock	unlock the FMC bank0 operation
fmc_bank1_unlock	unlock the FMC bank1 operation
fmc_lock	lock the main FMC operation
fmc_bank0_lock	lock the FMC bank0 operation
fmc_bank1_lock	lock the FMC bank1 operation
fmc_page_erase	erase page
fmc_mass_erase	erase whole chip
fmc_bank0_erase	erase bank0
fmc_bank1_erase	erase bank1
fmc_word_program	program a word at the corresponding address
fmc_halfword_program	program a half word at the corresponding address
fmc_otp_word_program	program a word at the corresponding address
fmc_otp_half_word_program	program a half word at the corresponding address
fmc_otp_byte_program	program a byte at the corresponding address
fmc_nwa_enable	enable no waiting time area load after system reset

fmc_nwa_disable	disable no waiting time area load after system reset
otp1_read_disable	set OTP1 data block not be read
otp2_rlock_enable	enable read lock block protection for OTP2
fmc_deep_power_down_enable	enable deep power down mode when no operation
fmc_deep_power_down_disable	disable deep power down mode when no operation
ob_unlock	unlock the option byte operation
ob_lock	lock the option byte operation
ob_start	send option byte change command
ob_erase	erase option byte
ob_write_protection_enable	enable write protection
ob_write_protection_disable	disable write protection
ob_security_protection_config	configure security protection level
ob_user_write	program the FMC user option byte
ob_sram_init_config	configure the option byte sramc initialize value
ob_sram_ecc_config	configure the option byte sram ecc function value
ob_nwa_clock_config	configure the option byte the clock for loading no waiting time area
ob_data_write	program option bytes data
ob_sram_init_get	get the option byte sram initialize value
ob_sram_ecc_get	get the option byte sramc ecc enable value
ob_nwa_clock_get	get the option byte the clock for loading no waiting time area
ob_data_get	get the FMC data option byte
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_interrupt_flag_get	get FMC interrupt flag state
fmc_interrupt_flag_clear	clear FMC interrupt flag state
fmc_flag_get	check flag is set or not
fmc_flag_clear	clear the FMC flag
fmc_bank0_state_get	get the FMC bank0 state
fmc_bank1_state_get	get the FMC bank1 state
fmc_bank0_ready_wait	check whether FMC bank0 is ready or not
fmc_bank1_ready_wait	check whether FMC bank1 is ready or not

## Enum fmc\_state\_enum

**Table 3-326. Enum fmc\_state\_enum**

Member name	Descriptions
FMC_READY	the operation has been completed
FMC_BUSY	the operation is in progress
FMC_PGERR	program error
FMC_WPERR	erase/program protection error

FMC_TOERR	timeout error
-----------	---------------

## Enum fmc\_interrupt\_enum

**Table 3-327. Enum fmc\_interrupt\_enum**

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

## Enum fmc\_flag\_enum

**Table 3-328. Enum fmc\_flag\_enum**

Member name	Descriptions
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 3-329. Enum fmc\_interrupt\_flag\_enum**

Member name	Descriptions
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
FMC_INT_FLAG_BANK1_END	FMC bank1 end of operation interrupt flag bit

## fmc\_unlock

The description of fmc\_unlock is shown as below:

**Table 3-330. Function fmc\_unlock**

Function name	fmc_unlock
Function prototype	void fmc_unlock(void)
Function descriptions	unlock the main FMC operation
Precondition	-

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

Example:

```
/* unlock fmc */
```

```
fmc_unlock();
```

### fmc\_bank0\_unlock

The description of fmc\_bank0\_unlock is shown as below:

**Table 3-331. Function fmc\_bank0\_unlock**

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

Example:

```
/* unlock fmc bank0 */
```

```
fmc_bank0_unlock();
```

### fmc\_bank1\_unlock

The description of fmc\_bank1\_unlock is shown as below:

**Table 3-332. Function fmc\_bank1\_unlock**

Function name	fmc_bank1_unlock
Function prototype	void fmc_bank1_unlock(void)
Function descriptions	unlock the FMC bank1 operation
Precondition	-
The called functions	-
Input parameter{in}	

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

Example:

```
/* unlock fmc bank1 */
fmc_bank1_unlock();
```

### fmc\_lock

The description of fmc\_lock is shown as below:

**Table 3-333. 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>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock fmc */
fmc_lock();
```

### fmc\_bank0\_lock

The description of fmc\_bank0\_lock is shown as below:

**Table 3-334. 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 fmc bank0 */
```

```
fmc_bank0_lock();
```

### fmc\_bank1\_lock

The description of fmc\_bank1\_lock is shown as below:

**Table 3-335. Function fmc\_bank1\_lock**

<b>Function name</b>	fmc_bank1_lock
<b>Function prototype</b>	void fmc_bank1_lock(void)
<b>Function descriptions</b>	lock 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:

```
/* lock fmc bank1 */
```

```
fmc_bank1_lock();
```

### fmc\_page\_erase

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

**Table 3-336. Function fmc\_page\_erase**

<b>Function name</b>	fmc_page_erase
<b>Function prototype</b>	fmc_state_enum fmc_page_erase(uint32_t page_address)
<b>Function descriptions</b>	erase page
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>page_address</b>	the page address to be erased.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

fmc_state_enum	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>
----------------	--

Example:

```
/* erase page of 0x08000000 */

fmc_state_enum state;

state = fmc_page_erase(0x08000000);
```

### fmc\_mass\_erase

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

**Table 3-337. 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>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* erase whole chip */

fmc_state_enum state;

state = fmc_mass_erase();
```

### fmc\_bank0\_erase

The description of fmc\_bank0\_erase is shown as below:

**Table 3-338. 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>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-



Return value	
fmc_state_enum	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* erase bank0 */

fmc_state_enum state;

state = fmc_bank0_erase();
```

### fmc\_bank1\_erase

The description of fmc\_bank1\_erase is shown as below:

**Table 3-339. 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>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* erase bank1 */

fmc_state_enum state;

state = fmc_bank1_erase();
```

### fmc\_word\_program

The description of fmc\_word\_program is shown as below:

**Table 3-340. 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>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
address	address to program
<b>Input parameter{in}</b>	

<b>data</b>	word to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* Write 0x12345678 to the address 0x08000000. */
```

```
fmc_state_enum state;
```

```
state = fmc_word_program(0x08000000, 0x12345678);
```

### fmc\_halfword\_program

The description of fmc\_halfword\_program is shown as below:

**Table 3-341. 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 half word at the corresponding address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>address</b>	address to program
<b>Input parameter{in}</b>	
<b>data</b>	halfword to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* Write 0x1234 to the address 0x08000000. */
```

```
fmc_state_enum state;
```

```
state = fmc_word_program(0x08000000, 0x1234);
```

### fmc\_otp\_word\_program

The description of fmc\_otp\_word\_program is shown as below:

**Table 3-342. Function fmc\_otp\_word\_program**

<b>Function name</b>	fmc_otp_word_program
<b>Function prototype</b>	fmc_state_enum fmc_otp_word_program(uint32_t address, uint32_t data)
<b>Function descriptions</b>	program a word at the corresponding address

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>address</b>	OTP address to program
<b>Input parameter{in}</b>	
<b>data</b>	word to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* Write 0x12345678 to the address 0x1FF00000 */

fmc_state_enum state;

state = fmc_otp_word_program(0x1FF00000, 0x12345678);
```

### fmc\_otp\_half\_word\_program

The description of fmc\_otp\_half\_word\_program is shown as below:

**Table 3-343. Function fmc\_otp\_half\_word\_program**

<b>Function name</b>	fmc_otp_half_word_program
<b>Function prototype</b>	fmc_state_enum fmc_otp_half_word_program(uint32_t address, uint16_t data)
<b>Function descriptions</b>	program a half word at the corresponding address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>address</b>	OTP address to program
<b>Input parameter{in}</b>	
<b>data</b>	halfword to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* Write 0x1234 to the address 0x1FF00000 */

fmc_state_enum state;

state = fmc_otp_half_word_program(0x1FF00000, 0x1234);
```

## fmc\_otp\_byte\_program

The description of fmc\_otp\_byte\_program is shown as below:

**Table 3-344. Function fmc\_otp\_byte\_program**

<b>Function name</b>	fmc_otp_byte_program
<b>Function prototype</b>	fmc_state_enum fmc_otp_byte_program(uint32_t address, uint8_t data)
<b>Function descriptions</b>	program a byte at the corresponding address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>address</b>	OTP address to program
<b>Input parameter{in}</b>	
<b>data</b>	byte to program(0x00 - 0xFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, refer to <a href="#">Enum fmc_state_enum</a>

Example:

```
/* Write 0x12 to the address 0x1FF00000 */

fmc_state_enum state;

state = fmc_otp_byte_program(0x1FF00000, 0x12);
```

## fmc\_nwa\_enable

The description of fmc\_nwa\_enable is shown as below:

**Table 3-345. Function fmc\_nwa\_enable**

<b>Function name</b>	fmc_nwa_enable
<b>Function prototype</b>	void fmc_nwa_enable(void)
<b>Function descriptions</b>	enable no waiting time area load after system reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable no waiting time area load after system reset */

fmc_nwa_enable();
```

## fmc\_nwa\_disable

The description of fmc\_nwa\_disable is shown as below:

**Table 3-346. Function fmc\_nwa\_disable**

<b>Function name</b>	fmc_nwa_disable
<b>Function prototype</b>	void fmc_nwa_disable(void)
<b>Function descriptions</b>	disable no waiting time area load after system reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable no waiting time area load after system reset */
fmc_nwa_disable();
```

## otp1\_read\_disable

The description of otp1\_read\_disable is shown as below:

**Table 3-347. Function otp1\_read\_disable**

<b>Function name</b>	otp1_read_disable
<b>Function prototype</b>	void otp1_read_disable(uint32_t block)
<b>Function descriptions</b>	set OTP1 data block not be read
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>block</b>	specify OTP1 data block x not be read
OTP1_DATA_BLOCK_x	data block x(x = 0,1,2...15)
OTP1_DATA_BLOCK_ALL	all data blocks
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set all OTP1 data block not be read */
otp1_read_disable(OTP1_DATA_BLOCK_ALL);
```

## otp2\_rlock\_enable

The description of otp2\_rlock\_enable is shown as below:

**Table 3-348. Function otp2\_rlock\_enable**

<b>Function name</b>	otp2_rlock_enable
<b>Function prototype</b>	void otp2_rlock_enable(void)
<b>Function descriptions</b>	enable read lock block protection for OTP2
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable read lock block protection for OTP2 */
otp2_rlock_enable();
```

## fmc\_deep\_power\_down\_enable

The description of fmc\_deep\_power\_down\_enable is shown as below:

**Table 3-349. Function fmc\_deep\_power\_down\_enable**

<b>Function name</b>	fmc_deep_power_down_enable
<b>Function prototype</b>	void fmc_deep_power_down_enable(void)
<b>Function descriptions</b>	enable deep power down mode when no operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable deep power down mode when no operation */
fmc_deep_power_down_enable();
```

## fmc\_deep\_power\_down\_disable

The description of fmc\_deep\_power\_down\_disable is shown as below:

**Table 3-350. Function fmc\_deep\_power\_down\_disable**

<b>Function name</b>	fmc_deep_power_down_disable
<b>Function prototype</b>	void fmc_deep_power_down_disable(void)
<b>Function descriptions</b>	disable deep power down mode when no operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable deep power down mode when no operation */
fmc_deep_power_down_disable();
```

### ob\_unlock

The description of ob\_unlock is shown as below:

**Table 3-351. Function ob\_unlock**

<b>Function name</b>	ob_unlock
<b>Function prototype</b>	void ob_unlock(void)
<b>Function descriptions</b>	unlock the option byte operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* unlock the option byte operation */
ob_unlock();
```

### ob\_lock

The description of ob\_lock is shown as below:

**Table 3-352. Function ob\_lock**

<b>Function name</b>	ob_lock
----------------------	---------

<b>Function prototype</b>	void ob_lock(void)
<b>Function descriptions</b>	lock the option byte operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock the option byte operation */
ob_lock();
```

### ob\_start

The description of ob\_start is shown as below:

**Table 3-353. Function ob\_start**

<b>Function name</b>	ob_start
<b>Function prototype</b>	void ob_start(void)
<b>Function descriptions</b>	send option byte change command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* send option byte change command */
ob_start();
```

### ob\_erase

The description of ob\_erase is shown as below:

**Table 3-354. Function ob\_erase**

<b>Function name</b>	ob_erase
<b>Function prototype</b>	void ob_erase(void)
<b>Function descriptions</b>	erase option byte



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

Example:

```
/* erase option byte */
```

```
ob_erase();
```

### ob\_write\_protection\_enable

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

**Table 3-355. Function ob\_write\_protection\_enable**

<b>Function name</b>	ob_write_protection_enable
<b>Function prototype</b>	ErrStatus ob_write_protection_enable(uint32_t ob_wp)
<b>Function descriptions</b>	enable write protection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ob_wp</b>	specify page to be write protected
<i>OB_WP_x(x=0..31)</i>	page x(x = 0,1,2...31)
<i>OB_WP_ALL</i>	all pages
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
ErrStatus	<b>SUCCESS or ERROR</b>

Example:

```
/* enable page0/1 write protection */
```

```
ob_write_protection_enable(OB_WP_0);
```

### ob\_write\_protection\_disable

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

**Table 3-356. Function ob\_write\_protection\_disable**

<b>Function name</b>	ob_write_protection_disable
<b>Function prototype</b>	ErrStatus ob_write_protection_disable(uint32_t ob_wp)
<b>Function descriptions</b>	disable write protection

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ob_wp</b>	specify sector to be write protected
<i>OB_WP_x(x=0..31)</i>	sector x(x = 0,1,2...31)
<i>OB_WP_ALL</i>	all sector
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
ErrStatus	<b>SUCCESS or ERROR</b>

Example:

```
/* disable page0/1 write protection */
ob_write_protection_disable(OB_WP_0);
```

### ob\_security\_protection\_config

The description of ob\_security\_protection\_config is shown as below:

**Table 3-357. Function ob\_security\_protection\_config**

<b>Function name</b>	ob_security_protection_config
<b>Function prototype</b>	void ob_security_protection_config(uint32_t ob_spc)
<b>Function descriptions</b>	configure security protection level
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ob_spc</b>	specify security protection level
<i>FMC_NSPC</i>	no security protection
<i>FMC_LSPC</i>	low security protection
<i>FMC_HSPC</i>	high security protection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure low security protection level */
ob_security_protection_config(FMC_LSPC);
```

### ob\_user\_write

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

Table 3-358. Function ob\_user\_write

Function name	ob_user_write
Function prototype	void ob_user_write(uint32_t ob_fwdgt, uint32_t ob_deepsleep, uint32_t ob_stdby)
Function descriptions	program the FMC user option byte
Precondition	-
The called functions	-
Input parameter{in}	
ob_fwdgt	option byte watchdog value
OB_FWDGT_SW	software free watchdog
OB_FWDGT_HW	hardware free watchdog
Input parameter{in}	
ob_deepsleep	option byte deepsleep reset value
OB_DEEPSLEEP_NRS T	no reset when entering deepsleep mode
OB_DEEPSLEEP_RST	generate a reset instead of entering deepsleep mode
Input parameter{in}	
ob_stdby	option byte standby reset value
OB_STDBY_NRS	no reset when entering standby mode
OB_STDBY_RST	generate a reset instead of entering standby mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* Configure the option bytes to enable the hardware watchdog */
```

```
ob_user_write(OB_FWDGT_HW, OB_DEEPSLEEP_NRS, OB_STDBY_NRS);
```

### ob\_sram\_init\_config

The description of ob\_sram\_init\_config is shown as below:

Table 3-359. Function ob\_sram\_init\_config

Function name	ob_sram_init_config
Function prototype	void ob_sram_init_config(uint32_t init_mode)
Function descriptions	configure the option byte sramc initialize value
Precondition	-
The called functions	-
Input parameter{in}	
init_mode	specifies the option byte sramc initialize value
OB_SRAMCINI_ENABLE	sramc initialize after power reset
OB_SRAMCINI_DISABLE	sramc no initialize after power reset
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* Enable SRAM power-on initialization */
```

```
ob_sram_init_config(OB_SRAMCINI_ENABLE);
```

### ob\_sram\_ecc\_config

The description of ob\_sram\_ecc\_config is shown as below:

**Table 3-360. Function ob\_sram\_ecc\_config**

<b>Function name</b>	ob_sram_ecc_config
<b>Function prototype</b>	void ob_sram_ecc_config(uint32_t ecc_mode)
<b>Function descriptions</b>	configure the option byte sram ecc function value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ecc_mode</b>	specifies the option byte sramc ecc function value
OB_SRAMCINI_DISABLE	SRAM ECC disable
OB_SRAMCINI_ENABLE	SRAM ECC enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* Enable SRAM ECC verification */
```

```
ob_sram_ecc_config(OB_SRAMCINI_ENABLE);
```

### ob\_nwa\_clock\_config

The description of ob\_nwa\_clock\_config is shown as below:

**Table 3-361. Function ob\_nwa\_clock\_config**

<b>Function name</b>	ob_nwa_clock_config
<b>Function prototype</b>	void ob_nwa_clock_config(uint32_t clock_mode)
<b>Function descriptions</b>	configure the option byte the clock for loading no waiting time area
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>clock_mode</b>	specifies the clock after Power reset when read flash to memory
PLL_CLK_200M	200M from PLL

<i>PLL_CLK_160M</i>	160M from PLL
<i>PLL_CLK_120M</i>	120M from PLL
<i>HSI_CLK_8M</i>	8M from HSI
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select PLL 200MHz */
```

```
ob_nwa_clock_config(PLL_CLK_200M);
```

### ob\_data\_write

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

**Table 3-362. Function ob\_data\_write**

<b>Function name</b>	ob_data_write
<b>Function prototype</b>	void ob_data_write(uint16_t data)
<b>Function descriptions</b>	program option bytes data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data</b>	data to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* Program the option byte DATA field to 0x1234 */
```

```
ob_data_write(0x1234);
```

### ob\_sram\_init\_get

The description of ob\_sram\_init\_get is shown as below:

**Table 3-363. Function ob\_sram\_init\_get**

<b>Function name</b>	ob_sram_init_get
<b>Function prototype</b>	uint32_t ob_sram_init_get(void)
<b>Function descriptions</b>	get the option byte sram initialize value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

-	-
Output parameter{out}	
-	-
Return value	
uint32_t	SRAM initialize value
OB_SRAMCINI_DISABLE	disable
OB_SRAMCINI_DISABLE	enable

Example:

```
/* Obtain the SRAM initialization configuration status. */
```

```
uint32_t sram_init;
```

```
sram_init = ob_sram_init_get();
```

### ob\_sram\_ecc\_get

The description of ob\_sram\_ecc\_get is shown as below:

**Table 3-364. Function ob\_sram\_ecc\_get**

Function name	ob_sram_ecc_get
Function prototype	uint32_t ob_sram_ecc_get(void)
Function descriptions	get the option byte sramc ecc enable value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	SRAM ECC function value
OB_SRAMCINI_ENABLE	enable SRAM ECC
OB_SRAMCINI_DISABLE	disable SRAM ECC

Example:

```
/* get the option byte sramc ecc enable value */
```

```
uint32_t sram_ecc;
```

```
sram_ecc = ob_sram_ecc_get();
```

### ob\_nwa\_clock\_get

The description of ob\_nwa\_clock\_get is shown as below:

**Table 3-365. Function ob\_nwa\_clock\_get**

Function name	ob_nwa_clock_get
---------------	------------------

<b>Function prototype</b>	uint32_t ob_nwa_clock_get(void)
<b>Function descriptions</b>	get the option byte the clock for loading no waiting time area
<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 option byte the clock for loading no waiting time area value
PLL_CLK_200M	200M from PLL
PLL_CLK_160M	160M from PLL
PLL_CLK_120M	120M from PLL
HSI_CLK_8M	8M from HSI

Example:

```
/* get the option byte the clock for loading no waiting time area */
```

```
uint32_t nwa_clock;
```

```
nwa_clock = ob_nwa_clock_get();
```

## ob\_data\_get

The description of ob\_data\_get is shown as below:

**Table 3-366. Function ob\_data\_get**

<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	Option byte DATA field value.

Example:

```
/* check whether FMC bank1 is ready or not */
```

```
uint16_t ob_data;
```

```
ob_data = ob_data_get();
```

## ob\_write\_protection\_get

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

**Table 3-367. 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 option byte write protection value

Example:

```
/* get the FMC option byte write protection */
uint32_t wp_value;

wp_value = ob_write_protection_get();
```

## ob\_spc\_get

The description of ob\_spc\_get is shown as below:

**Table 3-368. Function ob\_spc\_get**

<b>Function name</b>	ob_spc_get
<b>Function prototype</b>	uint32_t ob_spc_get(void)
<b>Function descriptions</b>	get option byte security protection code 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	FMC_NSPC, FMC_LSPC, FMC_HSPC

Example:

```
/* get option byte security protection code value */
uint32_t ob_spc;

ob_spc = ob_spc_get();
```



## fmc\_interrupt\_enable

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

**Table 3-369. Function fmc\_interrupt\_enable**

<b>Function name</b>	fmc_interrupt_enable
<b>Function prototype</b>	void fmc_interrupt_enable(fmc_interrupt_enum interrupt)
<b>Function descriptions</b>	enable FMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the FMC interrupt source
<i>FMC_INT_BANK0_END</i>	enable FMC bank0 end of operation
<i>FMC_INT_BANK0_ERR</i>	enable FMC bank0 error interrupt
<i>FMC_INT_BANK1_END</i>	enable FMC bank1 end of operation
<i>FMC_INT_BANK1_ERR</i>	enable FMC bank1 error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* Enable BANK0 complete interrupt. */
fmc_interrupt_enable(FMC_INT_BANK0_END);
```

## fmc\_interrupt\_disable

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

**Table 3-370. Function fmc\_interrupt\_disable**

<b>Function name</b>	fmc_interrupt_disable
<b>Function prototype</b>	void fmc_interrupt_disable(fmc_interrupt_enum interrupt)
<b>Function descriptions</b>	disable FMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the FMC interrupt source
<i>FMC_INT_BANK0_END</i>	enable FMC bank0 end of operation
<i>FMC_INT_BANK0_ERR</i>	enable FMC bank0 error interrupt
<i>FMC_INT_BANK1_END</i>	enable FMC bank1 end of operation
<i>FMC_INT_BANK1_ERR</i>	enable FMC bank1 error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* disable BANK0 complete interrupt. */
fmc_interrupt_disable(FMC_INT_BANK0_END);
```

### fmc\_interrupt\_flag\_get

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

**Table 3-371. 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 interrupt flags, refer to fmc_interrupt_flag_enum
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
FMC_INT_FLAG_BANK1_END	FMC bank1 end of operation interrupt flag bit
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* Get the BANK0 operation complete flag */
FlagStatus flag;
flag = fmc_interrupt_flag_get(FMC_INT_FLAG_BANK0_END);
```

### fmc\_interrupt\_flag\_clear

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

**Table 3-372. Function fmc\_interrupt\_flag\_clear**

Function name	fmc_interrupt_flag_clear
Function prototype	void fmc_interrupt_flag_clear(fmc_interrupt_flag_enum flag)
Function descriptions	clear FMC interrupt flag state
Precondition	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	FMC interrupt flags, refer to <code>fmc_interrupt_flag_enum</code>
<code>FMC_INT_FLAG_BANK0_PGERR</code>	FMC bank0 operation error interrupt flag bit
<code>FMC_INT_FLAG_BANK0_WPERR</code>	FMC bank0 erase/program protection error interrupt flag bit
<code>FMC_INT_FLAG_BANK0_END</code>	FMC bank0 end of operation interrupt flag bit
<code>FMC_INT_FLAG_BANK1_PGERR</code>	FMC bank1 operation error interrupt flag bit
<code>FMC_INT_FLAG_BANK1_WPERR</code>	FMC bank1 erase/program protection error interrupt flag bit
<code>FMC_INT_FLAG_BANK1_END</code>	FMC bank1 end of operation interrupt flag bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the BANK0 operation complete flag */
```

```
fmc_interrupt_flag_clear(FMC_INT_FLAG_BANK0_END);
```

### fmc\_flag\_get

The description of `fmc_flag_get` is shown as below:

**Table 3-373. Function `fmc_flag_get`**

<b>Function name</b>	<code>fmc_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus fmc_flag_get(fmc_flag_enum flag)</code>
<b>Function descriptions</b>	check flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	check FMC flag
<code>FMC_FLAG_BANK0_BUSY</code>	FMC bank0 busy flag bit
<code>FMC_FLAG_BANK0_PGERR</code>	FMC bank0 operation error flag bit
<code>FMC_FLAG_BANK0_WPERR</code>	FMC bank0 erase/program protection error flag bit
<code>FMC_FLAG_BANK0_END</code>	FMC bank0 end of operation flag bit
<code>FMC_FLAG_OBERR</code>	FMC option bytes read error flag bit
<code>FMC_FLAG_BANK1_BUSY</code>	FMC bank1 busy flag bit
<code>FMC_FLAG_BANK1_PGERR</code>	FMC bank1 operation error flag bit
<code>FMC_FLAG_BANK1_WPERR</code>	FMC bank1 erase/program protection error flag bit
<code>FMC_FLAG_BANK1_END</code>	FMC bank1 end of operation flag bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* Get the BANK0 complete interrupt flag. */
```

```
FlagStatus flag;
```

```
flag = fmc_flag_get(FMC_FLAG_BANK0_END);
```

### fmc\_flag\_clear

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

**Table 3-374. Function fmc\_flag\_clear**

Function name	fmc_flag_clear
Function prototype	void fmc_flag_clear(fmc_flag_enum flag)
Function descriptions	clear the FMC flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	clear FMC 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_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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the BANK0 complete interrupt flag */
```

```
FlagStatus flag;
```

```
flag = fmc_flag_clear(FMC_FLAG_BANK0_END);
```

### fmc\_bank0\_state\_get

The description of fmc\_bank0\_state\_get is shown as below:

**Table 3-375. 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 BANK0 status. */

fmc_state_enum state;

state = fmc_bank0_state_get();
```

### fmc\_bank1\_state\_get

The description of fmc\_bank1\_state\_get is shown as below:

**Table 3-376. 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 BANK1 status. */

fmc_state_enum state;

state = fmc_bank1_state_get();
```

### fmc\_bank0\_ready\_wait

The description of fmc\_bank0\_ready\_wait is shown as below:

**Table 3-377. 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	-
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;

state = fmc_bank0_ready_wait(FMC_TIMEOUT_COUNT);
```

### fmc\_bank1\_ready\_wait

The description of fmc\_bank1\_ready\_wait is shown as below:

**Table 3-378. Function fmc\_bank1\_ready\_wait**

Function name	fmc_bank1_ready_wait
Function prototype	fmc_state_enum fmc_bank1_ready_wait(uint32_t timeout)
Function descriptions	check whether FMC bank1 is ready or not
Precondition	-
The called functions	-
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 bank1 is ready or not */

fmc_state_enum state;

state = fmc_bank1_ready_wait(FMC_TIMEOUT_COUNT);
```

## 3.15. FWDGT

The free watchdog timer (FWDGT) is a hardware timing circuitry that can be used to detect system failures due to software malfunctions. It's suitable for the situation that requires an independent environment and lower timing accuracy. The FWDGT registers are listed in chapter [3.15.1](#) the FWDGT firmware functions are introduced in chapter [3.15.2](#).

### 3.15.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

**Table 3-379. FWDGT Registers**

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

### 3.15.2. Descriptions of Peripheral functions

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

**Table 3-380. FWDGT firmware function**

Function name	Function description
fwdgt_write_enable	enable write access to FWDGT_PSC and FWDGT_RLD
fwdgt_write_disable	disable write access to FWDGT_PSC and FWDGT_RLD
fwdgt_enable	start the free watchdog timer counter
fwdgt_prescaler_value_config	configure the free watchdog timer counter prescaler value
fwdgt_reload_value_config	configure the free watchdog timer counter reload value
fwdgt_counter_reload	reload the counter of FWDGT
fwdgt_config	configure counter reload value, and prescaler divider value
fwdgt_flag_get	get flag state of FWDGT

#### fwdgt\_write\_enable

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

**Table 3-381. 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 3-382. 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 3-383. 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>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* start the free watchdog timer counter */

```

```

fwdgt_enable( );

```



## fwdgt\_prescaler\_value\_config

The description of fwdgt\_prescaler\_value\_config is shown as below:

**Table 3-384. Function fwdgt\_prescaler\_value\_config**

<b>Function name</b>	fwdgt_prescaler_value_config
<b>Function prototype</b>	ErrStatus fwdgt_prescaler_value_config(uint16_t prescaler_value);
<b>Function descriptions</b>	configure the free watchdog timer counter prescaler value
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>prescaler_value</b>	specify prescaler value
<i>FWDGT_PSC_DIV4</i>	FWDGT prescaler set to 4
<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16
<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32
<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64
<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
<i>FWDGT_PSC_DIV512</i>	FWDGT prescaler set to 512
<i>FWDGT_PSC_DIV1024</i>	FWDGT prescaler set to 1024
<i>4</i>	
<i>FWDGT_PSC_DIV2048</i>	FWDGT prescaler set to 2048
<i>8</i>	
<i>FWDGT_PSC_DIV4096</i>	FWDGT prescaler set to 4096
<i>6</i>	
<i>FWDGT_PSC_DIV8192</i>	FWDGT prescaler set to 8192
<i>2</i>	
<i>FWDGT_PSC_DIV16384</i>	FWDGT prescaler set to 16384
<i>84</i>	
<i>FWDGT_PSC_DIV32768</i>	FWDGT prescaler set to 32768
<i>68</i>	
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR / SUCCESS

Example:

```
/* set FWDGT prescaler to 256 */
```

```
ErrStatus flag;
```

```
flag = fwdgt_prescaler_value_config(FWDGT_PSC_DIV256);
```

## fwdgt\_reload\_value\_config

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

**Table 3-385. Function fwdgt\_reload\_value\_config**

<b>Function name</b>	fwdgt_reload_value_config
<b>Function prototype</b>	ErrStatus fwdgt_reload_value_config(uint16_t reload_value);
<b>Function descriptions</b>	configure the free watchdog timer counter reload value
<b>Precondition</b>	-
<b>Input parameter{in}</b>	
<b>reload_value</b>	reload_value: specify window value(0x0000 - 0x0FFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR / SUCCESS

Example:

```
/* set FWDGT reload value to 0x0FFF */
```

ErrStatus flag;

```
flag = fwdgt_reload_value_config(0x0FFF);
```

### fwdgt\_counter\_reload

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

**Table 3-386. Function fwdgt\_counter\_reload**

<b>Function name</b>	fwdgt_counter_reload
<b>Function prototype</b>	void fwdgt_counter_reload(void);
<b>Function descriptions</b>	reload the counter of FWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reload FWDGT counter */
```

```
fwdgt_counter_reload();
```

### fwdgt\_config

The description of fwdgt\_config is shown as below:

**Table 3-387. Function fwdgt\_config**

<b>Function name</b>	fwdgt_config
----------------------	--------------

<b>Function prototype</b>	ErrStatus fwdgt_config(uint16_t reload_value, uint8_t prescaler_div);
<b>Function descriptions</b>	configure counter reload value, and prescaler divider value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>reload_value</b>	specify reload value(0x0000 - 0x0FFF)
<b>Input parameter{in}</b>	
<b>prescaler_div</b>	FWDGT prescaler value
<i>FWDGT_PSC_DIV4</i>	FWDGT prescaler set to 4
<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16
<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32
<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64
<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
<i>FWDGT_PSC_DIV512</i>	FWDGT prescaler set to 512
<i>FWDGT_PSC_DIV1024</i>	FWDGT prescaler set to 1024
<i>4</i>	
<i>FWDGT_PSC_DIV2048</i>	FWDGT prescaler set to 2048
<i>8</i>	
<i>FWDGT_PSC_DIV4096</i>	FWDGT prescaler set to 4096
<i>6</i>	
<i>FWDGT_PSC_DIV8192</i>	FWDGT prescaler set to 8192
<i>2</i>	
<i>FWDGT_PSC_DIV16384</i>	FWDGT prescaler set to 16384
<i>84</i>	
<i>FWDGT_PSC_DIV32768</i>	FWDGT prescaler set to 32768
<i>68</i>	
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* configure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
```

```
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

## fwdgt\_flag\_get

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

**Table 3-388. 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
```

```
{
```

```
...
```

```
}
```

## 3.16. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.16.1](#), the GPIO firmware functions are introduced in chapter [3.16.2](#).

### 3.16.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

**Table 3-389. GPIO Registers**

Registers	Descriptions
GPIOx_CTL	GPIO port control register
GPIOx_OMODE	GPIO port output mode register
GPIOx_OSPD0	GPIO port output speed register 0
GPIOx_PUD	GPIO port pull-up/pull-down register

Registers	Descriptions
GPIOx_ISTAT	GPIO port input status register
GPIOx_OCTL	GPIO port output control register
GPIOx_BOP	GPIO port bit operation register
GPIOx_LOCK	GPIO port configuration lock register
GPIOx_AFSEL0	GPIO alternate function selected register 0
GPIOx_AFSEL1	GPIO alternate function selected register 1
GPIOx_BC	GPIO bit clear register
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

### 3.16.2. Descriptions of Peripheral functions

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

**Table 3-390. GPIO firmware function**

Function name	Function description
gpio_deinit	reset GPIO port
gpio_afio_deinit	reset alternate function I/O(AFIO)
gpio_mode_set	set GPIO mode
gpio_output_options_set	set GPIO output type and speed
gpio_bit_set	set GPIO pin bit
gpio_bit_reset	reset GPIO pin bit
gpio_bit_write	write data to the specified GPIO pin
gpio_port_write	write data to the specified GPIO port
gpio_input_bit_get	get GPIO pin input status
gpio_input_port_get	get GPIO port input status
gpio_output_bit_get	get GPIO pin output status
gpio_output_port_get	get GPIO port output status
gpio_af_set	set GPIO alternate function
gpio_exti_source_select	select GPIO pin exti sources
gpio_pin_lock	lock GPIO pin

#### gpio\_deinit

The description of gpio\_deinit is shown as below:

**Table 3-391. Function gpio\_deinit**

Function name	gpio_deinit
Function prototype	void gpio_deinit(uint32_t gpio_periph);
Function descriptions	reset GPIO port
Precondition	-

<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)
<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 3-392. 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\_mode\_set

The description of gpio\_mode\_set is shown as below:

**Table 3-393. Function gpio\_mode\_set**

<b>Function name</b>	gpio_mode_set
<b>Function prototype</b>	void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin);
<b>Function descriptions</b>	set GPIO mode
<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)
<b>Input parameter{in}</b>	
<b>mode</b>	gpio pin mode
<i>GPIO_MODE_INPUT</i>	input mode
<i>GPIO_MODE_OUTPUT</i> <i>T</i>	output mode
<i>GPIO_MODE_AF</i>	alternate function mode
<i>GPIO_MODE_ANALOG</i> <i>G</i>	analog mode
<b>Input parameter{in}</b>	
<b>pull_up_down</b>	gpio pin with pull-up or pull-down resistor
<i>GPIO_PUPD_NONE</i>	floating mode, no pull-up and pull-down resistors
<i>GPIO_PUPD_PULLUP</i>	with pull-up resistor
<i>GPIO_PUPD_PULLDOWN</i> <i>WN</i>	with pull-down resistor
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config PA0 as input mode with pullup */
```

```
gpio_mode_set (GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_PULLUP, GPIO_PIN_0);
```

### gpio\_output\_options\_set

The description of gpio\_output\_options\_set is shown as below:

**Table 3-394. Function gpio\_output\_options\_set**

<b>Function name</b>	gpio_output_options_set
<b>Function prototype</b>	void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);
<b>Function descriptions</b>	set GPIO output type and speed
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port

<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<b>Input parameter{in}</b>	
<b>otype</b>	gpio pin output mode
<i>GPIO_OTYPE_PP</i>	push pull mode
<i>GPIO_OTYPE_OD</i>	open drain mode
<b>Input parameter{in}</b>	
<b>speed</b>	gpio pin output max speed
<i>GPIO_OSPEED_LEVE L0</i>	output max speed level 0
<i>GPIO_OSPEED_LEVE L1</i>	output max speed level 1
<i>GPIO_OSPEED_LEVE L2</i>	output max speed level 2
<i>GPIO_OSPEED_LEVE L3</i>	output max speed level 3
<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:

```
/* config PA0 as push pull mode */
```

```
gpio_output_options_set (GPIOA, GPIO_OTYPE_PP, GPIO_OSPEED_2MHZ, GPIO_PIN_0);
```

## gpio\_bit\_set

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

**Table 3-395. Function gpio\_bit\_set**

<b>Function name</b>	gpio_bit_set
<b>Function prototype</b>	void gpio_bit_set(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	set GPIO pin bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<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 3-396. 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)
<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 3-397. 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)
<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 3-398. 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>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<b>Input parameter{in}</b>	
<b>data</b>	specify the value to be written to the port output data register
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-399. Function gpio\_input\_bit\_get**

<b>Function name</b>	gpio_input_bit_get
<b>Function prototype</b>	FlagStatus gpio_input_bit_get(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	get GPIO pin input status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get status of PA0 */
```

```
FlagStatus bit_state = gpio_input_bit_get (GPIOA, GPIO_PIN_0);
```

## gpio\_input\_port\_get

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

**Table 3-400. Function gpio\_input\_port\_get**

<b>Function name</b>	gpio_input_port_get
<b>Function prototype</b>	uint16_t gpio_input_port_get(uint32_t gpio_periph);
<b>Function descriptions</b>	get GPIO all pins input status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<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 3-401. Function gpio\_output\_bit\_get**

<b>Function name</b>	gpio_output_bit_get
<b>Function prototype</b>	FlagStatus gpio_output_bit_get(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	get GPIO pin output status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get output status of PA0 */

FlagStatus bit_state;

bit_state = gpio_output_bit_get (GPIOA, GPIO_PIN_0);
```

## gpio\_output\_port\_get

The description of gpio\_output\_port\_get is shown as below:

**Table 3-402. Function gpio\_output\_port\_get**

<b>Function name</b>	gpio_output_port_get
<b>Function prototype</b>	uint16_t gpio_output_port_get(uint32_t gpio_periph);
<b>Function descriptions</b>	get GPIO all pins output status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,C,D,E)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	0x0000-0xFFFF

Example:

```
/* get output value of Port A */
```

```
uint16_t port_state;
```

```
port_state = gpio_output_port_get (GPIOA);
```

### gpio\_af\_set

The description of gpio\_af\_set is shown as below:

**Table 3-403. Function gpio\_af\_set**

<b>Function name</b>	gpio_af_set
<b>Function prototype</b>	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);
<b>Function descriptions</b>	set GPIO alternate function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx(x = A,B,F)
<b>Input parameter{in}</b>	
<b>alt_func_num</b>	GPIO pin af function, please refer to specific device datasheet
<i>GPIO_AF_0</i>	alternate function 0
<i>GPIO_AF_1</i>	alternate function 1
<i>GPIO_AF_2</i>	alternate function 2
<i>GPIO_AF_3</i>	alternate function 3
<i>GPIO_AF_4</i>	alternate function 4
<i>GPIO_AF_5</i>	alternate function 5
<i>GPIO_AF_6</i>	alternate function 6
<i>GPIO_AF_7</i>	alternate function 7
<i>GPIO_AF_8</i>	alternate function 8
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/*set PA0 alternate function 0*/
```

```
gpio_af_set(GPIOA, GPIO_AF_0, GPIO_PIN_0);
```

## gpio\_exti\_source\_select

The description of gpio\_exti\_source\_select is shown as below:

**Table 3-404. 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)
<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\_pin\_lock

The description of gpio\_pin\_lock is shown as below:

**Table 3-405. Function gpio\_pin\_lock**

<b>Function name</b>	gpio_pin_lock
<b>Function prototype</b>	void gpio_pin_lock(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	lock GPIO pin bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port

<i>GPIOx</i>	GPIOx(x = A,B)
<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);
```

## 3.17. HAU

The HASH Acceleration Unit supports acceleration of SHA-256 algorithm. The HAU registers are listed in chapter [3.17.1](#). the HAU firmware functions are introduced in chapter [3.17.2](#).

### 3.17.1. Descriptions of Peripheral registers

HAU registers are listed in the table shown as below:

**Table 3-406. HAU Registers**

Registers	Descriptions
HAU_CTL	control register
HAU_DI	data input register
HAU_CFG	configuration register
HAU_DO0	data output register 0
HAU_DO1	data output register 1
HAU_DO2	data output register 2
HAU_DO3	data output register 3
HAU_DO4	data output register 4
HAU_DO5	data output register 5
HAU_DO6	data output register 6
HAU_DO7	data output register 7
HAU_INTEN	interrupt enable register
HAU_STAT	status and interrupt flag register

### 3.17.2. Descriptions of Peripheral functions

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

Table 3-407. HAU firmware function

Function name	Function description
hau_deinit	reset the HAU peripheral
hau_init	initialize the HAU peripheral parameters
hau_reset	reset the HAU processor core
hau_last_word_validbits_num_config	configure the number of valid bits in last word of the message
hau_data_write	write data to the IN FIFO
hau_infifo_words_num_get	return the number of words already written into the IN FIFO
hau_digest_read	read the message digest result
hau_digest_calculation_enable	enable digest calculation
hau_multiple_single_dma_config	configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer or not
hau_dma_enable	enable the HAU DMA interface
hau_dma_disable	disable the HAU DMA interface
hau_hash_sha_256	calculate digest using SHA256 in HASH mode
hau_flag_get	get the HAU flag status
hau_flag_clear	clear the HAU flag status
hau_interrupt_enable	enable the HAU interrupts
hau_interrupt_disable	disable the HAU interrupts
hau_interrupt_flag_get	get the HAU interrupt flag status
hau_interrupt_flag_clear	clear the HAU interrupt flag status

### Structure hau\_digest\_parameter\_struct

Table 3-408. Structure hau\_digest\_parameter\_struct

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

### hau\_deinit

The description of hau\_deinit is shown as below:

Table 3-409. Function hau\_deinit

Function name	hau_deinit
Function prototype	void hau_deinit(void);
Function descriptions	reset the HAU peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-



Example:

```
/* reset the HAU peripheral */
hau_deinit();
```

## hau\_init

The description of hau\_init is shown as below:

**Table 3-410. Function hau\_init**

<b>Function name</b>	hau_init
<b>Function prototype</b>	void hau_init(uint32_t datatype);
<b>Function descriptions</b>	initialize the HAU peripheral parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>datatype</b>	the HAU peripheral parameters
HAU_SWAPPING_32BIT	no swapping
HAU_SWAPPING_16BIT	half-word swapping
HAU_SWAPPING_8BIT	bytes swapping
HAU_SWAPPING_1BIT	bit swapping
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the HAU peripheral parameters */
hau_init(HAU_SWAPPING_8BIT);
```

## hau\_reset

The description of hau\_reset is shown as below:

**Table 3-411. Function hau\_reset**

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

-	-
Return value	
-	-

Example:

```
/* reset the HAU processor core */
```

```
hau_reset();
```

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

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

**Table 3-412. Function hau\_last\_word\_validbits\_num\_config**

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

Example:

```
/* configure the number of valid bits in last word of the message */
```

```
hau_last_word_validbits_num_config(0x10);
```

### hau\_data\_write

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

**Table 3-413. Function hau\_data\_write**

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

-	-
---	---

Example:

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

```
hau_data_write(0x10);
```

### hau\_infifo\_words\_num\_get

The description of hau\_infifo\_words\_num\_get is shown as below:

**Table 3-414. Function hau\_infifo\_words\_num\_get**

<b>Function name</b>	hau_infifo_words_num_get
<b>Function prototype</b>	uint32_t hau_infifo_words_num_get(void);
<b>Function descriptions</b>	return the number of words already written into the IN FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	0x0 – 0xFFFFFFFF

Example:

```
/* return the number of words already written into the IN FIFO */
```

```
uint32_t num;
```

```
num = hau_infifo_words_num_get();
```

### hau\_digest\_read

The description of hau\_digest\_read is shown as below:

**Table 3-415. Function hau\_digest\_read**

<b>Function name</b>	hau_digest_read
<b>Function prototype</b>	void hau_digest_read(hau_digest_parameter_struct* digestpara);
<b>Function descriptions</b>	read the message digest result
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
digestpara	the message digest result, refer to structure <a href="#">Table 3-408. Structure hau_digest_parameter_struct</a>
<b>Return value</b>	

-	-
---	---

Example:

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

### hau\_digest\_calculation\_enable

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

**Table 3-416. Function hau\_digest\_calculation\_enable**

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

Example:

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

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

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

**Table 3-417. Function hau\_multiple\_single\_dma\_config**

<b>Function name</b>	hau_multiple_single_dma_config
<b>Function prototype</b>	void hau_multiple_single_dma_config(uint32_t multi_single);
<b>Function descriptions</b>	configure single or multiple DMA is used, and digest calculation at the end of a DMA transfer or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>multi_single</b>	Multiple or single
<i>SINGLE_DMA_AUTO_DIGEST</i>	message padding and message digest calculation at the end of a DMA transfer
<i>MULTIPLE_DMA_NO_</i>	multiple DMA transfers needed and CALEN bit is not automatically set at

<i>DIGEST</i>	the end of a DMA transfer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer
or not */
```

```
hau_multiple_single_dma_config(SINGLE_DMA_AUTO_DIGEST);
```

### hau\_dma\_enable

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

**Table 3-418. Function hau\_dma\_enable**

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

Example:

```
/* enable the HAU DMA interface */
```

```
hau_dma_enable();
```

### hau\_dma\_disable

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

**Table 3-419. Function hau\_dma\_disable**

<b>Function name</b>	hau_dma_disable
<b>Function prototype</b>	void hau_dma_disable(void);
<b>Function descriptions</b>	disable the HAU DMA interface
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the HAU DMA interface */
```

```
hau_dma_disable();
```

## hau\_hash\_sha\_256

The description of hau\_hash\_sha\_256 is shown as below:

**Table 3-420. Function hau\_hash\_sha\_256**

Function name	hau_hash_sha_256
Function prototype	ErrStatus hau_hash_sha_256(uint8_t *input, uint32_t in_length, uint8_t output[32]);
Function descriptions	calculate digest using SHA256 in HASH mode
Precondition	-
The called functions	-
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA256 in HASH mode */
```

```
ErrStatus status = hau_hash_sha_256 (&input, 0x10, output[0]);
```

## hau\_flag\_get

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

**Table 3-421. Function hau\_flag\_get**

Function name	hau_flag_get
Function prototype	FlagStatus hau_flag_get(uint32_t flag);
Function descriptions	get the HAU flag status
Precondition	-
The called functions	-
Input parameter{in}	

flag	HAU flag status
<i>HAU_FLAG_DATA_INPUT</i>	there is enough space (16 bytes) in the input FIFO
<i>HAU_FLAG_CALCULATION_COMPLETE</i>	digest calculation is completed
<i>HAU_FLAG_DMA</i>	DMA is enabled (DMAE =1) or a transfer is processing
<i>HAU_FLAG_BUSY</i>	data block is in process
<i>HAU_FLAG_INFIFO_NO_EMPTY</i>	the input FIFO is not empty
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the HAU flag status */
```

```
FlagStatus status;
```

```
status = hau_flag_get (HAU_FLAG_DMA);
```

### hau\_flag\_clear

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

**Table 3-422. Function hau\_flag\_clear**

Function name	hau_flag_clear
Function prototype	void hau_flag_clear(uint32_t flag);
Function descriptions	clear the HAU flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	HAU flag status
<i>HAU_FLAG_DATA_INPUT</i>	there is enough space (16 bytes) in the input FIFO
<i>HAU_FLAG_CALCULATION_COMPLETE</i>	digest calculation is completed
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the HAU flag status */
```

```
hau_flag_clear (HAU_FLAG_DATA_INPUT);
```

## hau\_interrupt\_enable

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

**Table 3-423. Function hau\_interrupt\_enable**

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

Example:

```
/* enable hau interrupt */
hau_interrupt_enable(HAU_INT_DATA_INPUT);
```

## hau\_interrupt\_disable

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

**Table 3-424. Function hau\_interrupt\_disable**

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



-	-
---	---

Example:

```
/* disable hau interrupt */
hau_interrupt_disable(HAU_INT_DATA_INPUT);
```

## hau\_interrupt\_flag\_get

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

**Table 3-425. Function hau\_interrupt\_flag\_get**

<b>Function name</b>	hau_interrupt_flag_get
<b>Function prototype</b>	FlagStatus hau_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	get the HAU interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	HAU interrupt flag status
HAU_INT_FLAG_DATA_INPUT	there is enough space (16 bytes) in the input FIFO
HAU_INT_FLAG_CALCULATION_COMPLETE	digest calculation is completed
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the HAU interrupt flag status */
FlagStatus status = hau_interrupt_flag_get(HAU_INT_FLAG_DATA_INPUT);
```

## hau\_interrupt\_flag\_clear

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

**Table 3-426. Function hau\_interrupt\_flag\_clear**

<b>Function name</b>	hau_interrupt_flag_clear
<b>Function prototype</b>	void hau_interrupt_flag_clear(uint32_t int_flag)
<b>Function descriptions</b>	clear the HAU interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	HAU interrupt flag status
HAU_INT_FLAG_DATA	there is enough space (16 bytes) in the input FIFO

<code>_INPUT</code>	
<code>HAU_INT_FLAG_CALCULATION_COMPLETE</code>	digest calculation is completed
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the HAU interrupt flag status */
```

```
hau_interrupt_flag_clear(HAU_INT_FLAG_DATA_INPUT);
```

## 3.18. I2C

The I2C (inter-integrated circuit) module provides an I2C interface which is an industry standard two-line serial interface for MCU to communicate with external I2C interface. The I2C registers are listed in chapter [3.18.1](#), the I2C firmware functions are introduced in chapter [3.18.2](#).

### 3.18.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

**Table 3-427. I2C Registers**

Registers	Descriptions
I2C_CTL0	Control register 0
I2C_CTL1	Control register 1
I2C_SADDR0	Slave address register 0
I2C_SADDR1	Slave address register 1
I2C_TIMING	Timing register
I2C_TIMEOUT	Timeout register
I2C_STAT	Status register
I2C_STATC	I2C status clear register
I2C_PEC	PEC register
I2C_RDATA	Receive data register
I2C_TDATA	Transmit data register
I2C_CTL2	Control register 2

### 3.18.2. Descriptions of Peripheral functions

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

**Table 3-428. I2C firmware function**

Function name	Function description
i2c_deinit	reset I2C
i2c_timing_config	configure the timing parameters
i2c_digital_noise_filter_config	configure digital noise filter
i2c_analog_noise_filter_enable	enable analog noise filter
i2c_analog_noise_filter_disable	disable analog noise filter
i2c_master_clock_config	configure the SCL high and low period of clock in master mode
i2c_master_addressing	configure i2c slave addresss and transfer direction in master mode
i2c_address10_header_enable	10-bit address header executes read direction only in master receive mode
i2c_address10_header_disable	10-bit address header executes complete sequence in master receive mode
i2c_address10_enable	enable 10-bit addressing mode in master mode
i2c_address10_disable	disable 10-bit addressing mode in master mode
i2c_automatic_end_enable	enable I2C automatic end mode in master mode
i2c_automatic_end_disable	disable I2C automatic end mode in master mode
i2c_slave_response_to_gcall_enable	enable the response to a general call
i2c_slave_response_to_gcall_disable	disable the response to a general call
i2c_stretch_scl_low_enable	enable to stretch SCL low when data is not ready in slave mode
i2c_stretch_scl_low_disable	disable to stretch SCL low when data is not ready in slave mode
i2c_address_config	configure i2c slave address
i2c_address_bit_compare_config	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
i2c_address_disable	disable i2c address in slave mode
i2c_second_address_config	configure i2c second slave address
i2c_second_address_disable	disable i2c second address in slave mode
i2c_received_address_get	get received match address in slave mode
i2c_slave_byte_control_enable	enable slave byte control
i2c_slave_byte_control_disable	disable slave byte control
i2c_nack_enable	generate a NACK in slave mode
i2c_enable	enable I2C
i2c_disable	disable I2C
i2c_start_on_bus	generate a START condition on I2C bus
i2c_stop_on_bus	generate a STOP condition on I2C bus
i2c_data_transmit	I2C transmit data
i2c_data_receive	I2C receive data
i2c_reload_enable	enable I2C reload mode
i2c_reload_disable	disable I2C reload mode

Function name	Function description
i2c_transfer_byte_number_config	configure number of bytes to be transferred
i2c_dma_enable	enable I2C DMA for transmission or reception
i2c_dma_disable	disable I2C DMA for transmission or reception
i2c_pec_transfer	I2C transfers PEC value
i2c_pec_enable	enable I2C PEC calculation
i2c_pec_disable	disable I2C PEC calculation
i2c_pec_value_get	get packet error checking value
i2c_smbus_alert_enable	enable SMBus Alert
i2c_smbus_alert_disable	disable SMBus Alert
i2c_smbus_default_addr_enable	enable SMBus device default address
i2c_smbus_default_addr_disable	disable SMBus device default address
i2c_smbus_host_addr_enable	enable SMBus Host address
i2c_smbus_host_addr_disable	disable SMBus Host address
i2c_extended_clock_timeout_enable	enable extended clock timeout detection
i2c_extended_clock_timeout_disable	disable extended clock timeout detection
i2c_clock_timeout_enable	enable clock timeout detection
i2c_clock_timeout_disable	disable clock timeout detection
i2c_bus_timeout_b_config	configure bus timeout B
i2c_bus_timeout_a_config	configure bus timeout A
i2c_idle_clock_timeout_config	configure idle clock timeout detection
i2c_flag_get	get I2C flag status
i2c_flag_clear	clear I2C flag status
i2c_interrupt_enable	enable I2C interrupt
i2c_interrupt_disable	disable I2C interrupt
i2c_interrupt_flag_get	get I2C interrupt flag status
i2c_interrupt_flag_clear	clear I2C interrupt flag status

## Enum i2c\_interrupt\_flag\_enum

**Table 3-429. i2c\_interrupt\_flag\_enum**

Member name	Function description
I2C_INT_FLAG_TI	transmit interrupt flag
I2C_INT_FLAG_RBNE	I2C_RDATA is not empty during receiving interrupt flag
I2C_INT_FLAG_ADDSEND	address received matches in slave mode interrupt flag
I2C_INT_FLAG_NACK	not acknowledge interrupt flag
I2C_INT_FLAG_STPDET	stop condition detected in slave mode interrupt flag
I2C_INT_FLAG_TC	transfer complete in master mode interrupt flag
I2C_INT_FLAG_TCR	transfer complete reload interrupt flag
I2C_INT_FLAG_BERR	bus error interrupt flag
I2C_INT_FLAG_LOSTARB	arbitration lost interrupt flag
I2C_INT_FLAG_OUERR	overrun/underrun error in slave mode interrupt flag
I2C_INT_FLAG_PECERR	PEC error interrupt flag

Member name	Function description
I2C_INT_FLAG_TIMEOUT	timeout interrupt flag
I2C_INT_FLAG_SMBALT	SMBus Alert interrupt flag

## i2c\_deinit

The description of i2c\_deinit is shown as below:

**Table 3-430. Function i2c\_deinit**

<b>Function name</b>	i2c_deinit
<b>Function prototype</b>	void i2c_deinit(uint32_t i2c_periph);
<b>Function descriptions</b>	reset I2C
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset I2C0 */
i2c_deinit(I2C0);
```

## i2c\_timing\_config

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

**Table 3-431. Function i2c\_timing\_config**

<b>Function name</b>	i2c_timing_config
<b>Function prototype</b>	void i2c_timing_config(uint32_t i2c_periph, uint32_t psc, uint32_t scl_dely, uint32_t sda_dely);
<b>Function descriptions</b>	configure the timing parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>psc</b>	0-0x0000000F, timing prescaler
<b>Input parameter{in}</b>	
<b>scl_dely</b>	0-0x0000000F, data setup time
<b>Input parameter{in}</b>	

<b>sda_dely</b>	0-0x0000000F,data hold time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the timing parameters */
```

```
i2c_timing_config(I2C0, 0x1, 0x2, 0x1);
```

### i2c\_digital\_noise\_filter\_config

The description of i2c\_digital\_noise\_filter\_config is shown as below:

**Table 3-432. Function i2c\_digital\_noise\_filter\_config**

<b>Function name</b>	i2c_digital_noise_filter_config
<b>Function prototype</b>	void i2c_digital_noise_filter_config(uint32_t i2c_periph, uint32_t filter_length);
<b>Function descriptions</b>	configure digital noise filter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>filter_length</b>	filter_length
<i>FILTER_DISABLE</i>	digital filter is disabled
<i>FILTER_LENGTH_1</i>	digital filter is enabled and filter spikes with a length of up to 1 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_2</i>	digital filter is enabled and filter spikes with a length of up to 2 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_3</i>	digital filter is enabled and filter spikes with a length of up to 3 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_4</i>	digital filter is enabled and filter spikes with a length of up to 4 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_5</i>	digital filter is enabled and filter spikes with a length of up to 5 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_6</i>	digital filter is enabled and filter spikes with a length of up to 6 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_7</i>	digital filter is enabled and filter spikes with a length of up to 7 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_8</i>	digital filter is enabled and filter spikes with a length of up to 8 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_9</i>	digital filter is enabled and filter spikes with a length of up to 9 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_10</i>	digital filter is enabled and filter spikes with a length of up to 10 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_11</i>	digital filter is enabled and filter spikes with a length of up to 11 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_12</i>	digital filter is enabled and filter spikes with a length of up to 12 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_13</i>	digital filter is enabled and filter spikes with a length of up to 13 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_14</i>	digital filter is enabled and filter spikes with a length of up to 14 t <sub>I2CCLK</sub>
<i>FILTER_LENGTH_15</i>	digital filter is enabled and filter spikes with a length of up to 15 t <sub>I2CCLK</sub>
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* I2C0 digital filter filters spikes with a length of up to 1 tI2CCLK */
```

```
i2c_digital_noise_filter_config(I2C0, FILTER_LENGTH_1);
```

### i2c\_analog\_noise\_filter\_enable

The description of i2c\_analog\_noise\_filter\_enable is shown as below:

**Table 3-433. Function i2c\_analog\_noise\_filter\_enable**

Function name	i2c_analog_noise_filter_enable
Function prototype	void i2c_analog_noise_filter_enable(uint32_t i2c_periph);
Function descriptions	enable analog noise filter
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable analog noise filter */
```

```
i2c_analog_noise_filter_enable(I2C0);
```

### i2c\_analog\_noise\_filter\_disable

The description of i2c\_analog\_noise\_filter\_disable is shown as below:

**Table 3-434. Function i2c\_analog\_noise\_filter\_disable**

Function name	i2c_analog_noise_filter_disable
Function prototype	void i2c_analog_noise_filter_disable(uint32_t i2c_periph);
Function descriptions	disable analog noise filter
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable analog noise filter */
```

```
i2c_analog_noise_filter_disable(I2C0);
```

### i2c\_master\_clock\_config

The description of i2c\_master\_clock\_config is shown as below:

**Table 3-435. Function i2c\_master\_clock\_config**

Function name	i2c_master_clock_config
Function prototype	void i2c_master_clock_config(uint32_t i2c_periph, uint32_t sclh, uint32_t scll);
Function descriptions	configure the SCL high and low period of clock in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
sclh	0-0xff, SCL high period
Input parameter{in}	
scll	0-0xff, SCL low period
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the SCL and SDA period of clock in master mode */
```

```
i2c_master_clock_config(I2C0, 0x0f, 0x0f);
```

### i2c\_master\_addressing

The description of i2c\_master\_addressing is shown as below:

**Table 3-436. Function i2c\_master\_addressing**

Function name	i2c_master_addressing
Function prototype	void i2c_master_addressing(uint32_t i2c_periph, uint32_t address, uint32_t trans_direction);
Function descriptions	configure i2c slave addresss and transfer direction in master mode
Precondition	-



<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>address</b>	0-0x3FF except reserved address, I2C slave address to be sent
<b>Input parameter{in}</b>	
<b>trans_direction</b>	I2C transfer direction in master mode
<i>I2C_MASTER_TRANSMIT</i>	master transmit
<i>I2C_MASTER_RECEIVE</i>	master receive
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* send slave address to I2C bus */
```

```
i2c_master_addressing(I2C0, 0x82, I2C_MASTER_TRANSMIT);
```

### i2c\_address10\_header\_enable

The description of i2c\_address10\_header\_enable is shown as below:

**Table 3-437. Function i2c\_address10\_header\_enable**

<b>Function name</b>	i2c_address10_header_enable
<b>Function prototype</b>	void i2c_address10_header_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	10-bit address header executes read direction only in master receive mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 10-bit address header executes read direction only in master receive mode */
```

```
i2c_address10_header_enable(I2C0);
```

## i2c\_address10\_header\_disable

The description of i2c\_address10\_header\_disable is shown as below:

**Table 3-438. Function i2c\_address10\_header\_disable**

<b>Function name</b>	i2c_address10_header_disable
<b>Function prototype</b>	void i2c_address10_header_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	10-bit address header executes complete sequence in master receive mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 10-bit address header executes complete sequence in master receive mode */
i2c_address10_header_disable(I2C0);
```

## i2c\_address10\_enable

The description of i2c\_address10\_enable is shown as below:

**Table 3-439. Function i2c\_address10\_enable**

<b>Function name</b>	i2c_address10_enable
<b>Function prototype</b>	void i2c_address10_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable 10-bit addressing mode in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable 10-bit addressing mode in master mode */
i2c_address10_enable(I2C0);
```

## i2c\_address10\_disable

The description of i2c\_address10\_disable is shown as below:

**Table 3-440. Function i2c\_address10\_disable**

<b>Function name</b>	i2c_address10_disable
<b>Function prototype</b>	void i2c_address10_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable 10-bit addressing mode in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable 10-bit addressing mode in master mode */
```

```
i2c_address10_disable(I2C0);
```

## i2c\_automatic\_end\_enable

The description of i2c\_automatic\_end\_enable is shown as below:

**Table 3-441. Function i2c\_automatic\_end\_enable**

<b>Function name</b>	i2c_automatic_end_enable
<b>Function prototype</b>	void i2c_automatic_end_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable I2C automatic end mode in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_enable(I2C0);
```

## i2c\_automatic\_end\_disable

The description of i2c\_automatic\_end\_disable is shown as below:

**Table 3-442. Function i2c\_automatic\_end\_disable**

<b>Function name</b>	i2c_automatic_end_disable
<b>Function prototype</b>	void i2c_automatic_end_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable I2C automatic end mode in master mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_disable(I2C0);
```

## i2c\_slave\_response\_to\_gcall\_enable

The description of i2c\_slave\_response\_to\_gcall\_enable is shown as below:

**Table 3-443. Function i2c\_slave\_response\_to\_gcall\_enable**

<b>Function name</b>	i2c_slave_response_to_gcall_enable
<b>Function prototype</b>	void i2c_slave_response_to_gcall_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable the response to a general call
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the response to a general call */
```

```
i2c_slave_response_to_gcall_enable(I2C0);
```

## i2c\_slave\_response\_to\_gcall\_disable

The description of i2c\_slave\_response\_to\_gcall\_disable is shown as below:

**Table 3-444. Function i2c\_slave\_response\_to\_gcall\_disable**

<b>Function name</b>	i2c_slave_response_to_gcall_disable
<b>Function prototype</b>	void i2c_slave_response_to_gcall_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable the response to a general call
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the response to a general call */
i2c_slave_response_to_gcall_disable(I2C0);
```

## i2c\_stretch\_scl\_low\_enable

The description of i2c\_stretch\_scl\_low\_enable is shown as below:

**Table 3-445. Function i2c\_stretch\_scl\_low\_enable**

<b>Function name</b>	i2c_stretch_scl_low_enable
<b>Function prototype</b>	void i2c_stretch_scl_low_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable to stretch SCL low when data is not ready in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable to stretch SCL low when data is not ready in slave mode */
i2c_stretch_scl_low_enable(I2C0);
```

## i2c\_stretch\_scl\_low\_disable

The description of i2c\_stretch\_scl\_low\_disable is shown as below:

**Table 3-446. Function i2c\_stretch\_scl\_low\_disable**

<b>Function name</b>	i2c_stretch_scl_low_disable
<b>Function prototype</b>	void i2c_stretch_scl_low_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable to stretch SCL low when data is not ready in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_disable(I2C0);
```

## i2c\_address\_config

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

**Table 3-447. Function i2c\_address\_config**

<b>Function name</b>	i2c_address_config
<b>Function prototype</b>	void i2c_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_format);
<b>Function descriptions</b>	configure i2c slave address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>address</b>	I2C address
<b>Input parameter{in}</b>	
<b>addr_format</b>	7bits or 10bits
<i>I2C_ADDFORMAT_7BITS</i>	7bits
<i>I2C_ADDFORMAT_10BITS</i>	10bits

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure i2c slave address */
```

```
i2c_address_config(I2C0, 0x82, I2C_ADDFORMAT_7BITS);
```

### i2c\_address\_bit\_compare\_config

The description of i2c\_address\_bit\_compare\_config is shown as below:

**Table 3-448. Function i2c\_address\_bit\_compare\_config**

Function name	i2c_address_bit_compare_config
Function prototype	void i2c_address_bit_compare_config(uint32_t i2c_periph, uint32_t compare_bits);
Function descriptions	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
Precondition	-
The called functions	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
<b>compare_bits</b>	the bits need to compare
<i>ADDRESS_BIT1_COMPARE</i>	address bit1 needs compare
<i>ADDRESS_BIT2_COMPARE</i>	address bit2 needs compare
<i>ADDRESS_BIT3_COMPARE</i>	address bit3 needs compare
<i>ADDRESS_BIT4_COMPARE</i>	address bit4 needs compare
<i>ADDRESS_BIT5_COMPARE</i>	address bit5 needs compare
<i>ADDRESS_BIT6_COMPARE</i>	address bit6 needs compare
<i>ADDRESS_BIT7_COMPARE</i>	address bit7 needs compare
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* bit 1 of ADDRESS[7:1] need to compare with the incoming address byte */
i2c_address_bit_compare_config(I2C0, ADDRESS_BIT1_COMPARE);
```

### i2c\_address\_disable

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

**Table 3-449. Function i2c\_address\_disable**

<b>Function name</b>	i2c_address_disable
<b>Function prototype</b>	void i2c_address_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable i2c address in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable i2c address in slave mode */
i2c_address_disable(I2C0);
```

### i2c\_second\_address\_config

The description of i2c\_second\_address\_config is shown as below:

**Table 3-450. Function i2c\_second\_address\_config**

<b>Function name</b>	i2c_second_address_config
<b>Function prototype</b>	void i2c_second_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_mask);
<b>Function descriptions</b>	configure i2c second slave address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>address</b>	I2C address
<b>Input parameter{in}</b>	
<b>addr_mask</b>	the bits not need to compare



<code>ADDRESS2_NO_MASK</code> <code>K</code>	no mask, all the bits must be compared
<code>ADDRESS2_MASK_BIT1</code>	<code>ADDRESS2[1]</code> is masked, only <code>ADDRESS2[7:2]</code> are compared
<code>ADDRESS2_MASK_BIT1_2</code>	<code>ADDRESS2[2:1]</code> is masked, only <code>ADDRESS2[7:3]</code> are compared
<code>ADDRESS2_MASK_BIT1_3</code>	<code>ADDRESS2[3:1]</code> is masked, only <code>ADDRESS2[7:4]</code> are compared
<code>ADDRESS2_MASK_BIT1_4</code>	<code>ADDRESS2[4:1]</code> is masked, only <code>ADDRESS2[7:5]</code> are compared
<code>ADDRESS2_MASK_BIT1_5</code>	<code>ADDRESS2[5:1]</code> is masked, only <code>ADDRESS2[7:6]</code> are compared
<code>ADDRESS2_MASK_BIT1_6</code>	<code>ADDRESS2[6:1]</code> is masked, only <code>ADDRESS2[7]</code> are compared
<code>ADDRESS2_MASK_ALL</code> <code>L</code>	all the <code>ADDRESS2[7:1]</code> bits are masked
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure i2c second slave address */
```

```
i2c_second_address_config(I2C0, 0x82, ADDRESS2_MASK_BIT1_2);
```

### **i2c\_second\_address\_disable**

The description of `i2c_second_address_disable` is shown as below:

**Table 3-451. Function `i2c_second_address_disable`**

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

Example:

```
/* disable i2c second address in slave mode */
```

```
i2c_second_address_disable(I2C0);
```

### i2c\_received\_address\_get

The description of i2c\_received\_address\_get is shown as below:

**Table 3-452. Function i2c\_received\_address\_get**

<b>Function name</b>	i2c_received_address_get
<b>Function prototype</b>	uint32_t i2c_received_address_get(uint32_t i2c_periph);
<b>Function descriptions</b>	get received match address in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	0x00000000..0x0000007F

Example:

```
/* get received match address in slave mode */
```

```
uint32_t address;
```

```
address = i2c_received_address_get(I2C0);
```

### i2c\_slave\_byte\_control\_enable

The description of i2c\_slave\_byte\_control\_enable is shown as below:

**Table 3-453. Function i2c\_slave\_byte\_control\_enable**

<b>Function name</b>	i2c_slave_byte_control_enable
<b>Function prototype</b>	void i2c_slave_byte_control_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable slave byte control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable slave byte control */

i2c_slave_byte_control_enable(I2C0);
```

### i2c\_slave\_byte\_control\_disable

The description of i2c\_slave\_byte\_control\_disable is shown as below:

**Table 3-454. Function i2c\_slave\_byte\_control\_disable**

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

Example:

```
/* disable slave byte control */

i2c_slave_byte_control_disable(I2C0);
```

### i2c\_nack\_enable

The description of i2c\_nack\_enable is shown as below:

**Table 3-455. Function i2c\_nack\_enable**

<b>Function name</b>	i2c_nack_enable
<b>Function prototype</b>	void i2c_nack_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a NACK in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* generate a NACK in slave mode */
i2c_nack_enable(I2C0);
```

## i2c\_enable

The description of i2c\_enable is shown as below:

**Table 3-456. 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 3-457. 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 3-458. 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 3-459. 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 3-460. Function i2c\_data\_transmit**

<b>Function name</b>	i2c_data_transmit
<b>Function prototype</b>	void i2c_data_transmit(uint32_t i2c_periph, uint32_t data);
<b>Function descriptions</b>	I2C transmit data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>data</b>	transmit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 transmit data */
```

```
i2c_data_transmit(I2C0, 0x80);
```

### i2c\_data\_receive

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

**Table 3-461. Function i2c\_data\_receive**

<b>Function name</b>	i2c_data_receive
<b>Function prototype</b>	uint32_t i2c_data_receive(uint32_t i2c_periph);
<b>Function descriptions</b>	I2C receive data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

<b>uint32_t</b>	0x00000000..0x000000FF
-----------------	------------------------

Example:

```
/* I2C0 receive data */
```

```
uint32_t i2c_receiver;
```

```
i2c_receiver = i2c_data_receive(I2C0);
```

### i2c\_reload\_enable

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

**Table 3-462. Function i2c\_reload\_enable**

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

Example:

```
/* enable I2C reload mode */
```

```
i2c_reload_enable(I2C0);
```

### i2c\_reload\_disable

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

**Table 3-463. Function i2c\_reload\_disable**

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

Return value	
-	-

Example:

```
/* disable I2C reload mode */
```

```
i2c_reload_disable(I2C0);
```

### i2c\_transfer\_byte\_number\_config

The description of i2c\_transfer\_byte\_number\_config is shown as below:

**Table 3-464. Function i2c\_transfer\_byte\_number\_config**

Function name	i2c_transfer_byte_number_config
Function prototype	void i2c_transfer_byte_number_config(uint32_t i2c_periph, uint32_t byte_number);
Function descriptions	configure number of bytes to be transferred
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
byte_number	0x0-0xFF, number of bytes to be transferred
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure number of bytes to be transferred */
```

```
i2c_transfer_byte_number_config(I2C0, 0xFF);
```

### i2c\_dma\_enable

The description of i2c\_dma\_enable is shown as below:

**Table 3-465. Function i2c\_dma\_enable**

Function name	i2c_dma_enable
Function prototype	void i2c_dma_enable(uint32_t i2c_periph, uint32_t dma);
Function descriptions	enable I2C DMA for transmission or reception
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral



<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>dma</b>	I2C DMA
<i>I2C_DMA_TRANSMIT</i>	transmit data using DMA
<i>I2C_DMA_RECEIVE</i>	receive data using DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C DMA for transmission or reception */
```

```
i2c_dma_enable(I2C0, I2C_DMA_RECEIVE);
```

### i2c\_dma\_disable

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

**Table 3-466. Function i2c\_dma\_disable**

<b>Function name</b>	i2c_dma_disable
<b>Function prototype</b>	void i2c_dma_disable(uint32_t i2c_periph, uint32_t dma);
<b>Function descriptions</b>	disable I2C DMA for transmission or reception
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>dma</b>	I2C DMA
<i>I2C_DMA_TRANSMIT</i>	transmit data using DMA
<i>I2C_DMA_RECEIVE</i>	receive data using DMA
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C DMA for transmission or reception */
```

```
i2c_dma_disable(I2C0, I2C_DMA_RECEIVE);
```

### i2c\_pec\_transfer

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

Table 3-467. Function i2c\_pec\_transfer

Function name	i2c_pec_transfer
Function prototype	void i2c_pec_transfer(uint32_t i2c_periph);
Function descriptions	I2C transfers PEC value
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C transfers PEC value */
```

```
i2c_pec_transfer(I2C0);
```

### i2c\_pec\_enable

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

Table 3-468. Function i2c\_pec\_enable

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

Example:

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

```
i2c_pec_enable(I2C0);
```

### i2c\_pec\_disable

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

Table 3-469. Function i2c\_pec\_disable

Function name	i2c_pec_disable
Function prototype	void i2c_pec_disable(uint32_t i2c_periph);
Function descriptions	disable I2C PEC calculation
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
i2c_pec_disable(I2C0);
```

### i2c\_pec\_value\_get

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

Table 3-470. Function i2c\_pec\_value\_get

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

Example:

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

```
uint32_t pec_value;
```

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

## i2c\_smbus\_alert\_enable

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

**Table 3-471. Function i2c\_smbus\_alert\_enable**

<b>Function name</b>	i2c_smbus_alert_enable
<b>Function prototype</b>	void i2c_smbus_alert_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable SMBus Alert
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SMBus Alert */
i2c_smbus_alert_enable(I2C0);
```

## i2c\_smbus\_alert\_disable

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

**Table 3-472. Function i2c\_smbus\_alert\_disable**

<b>Function name</b>	i2c_smbus_alert_disable
<b>Function prototype</b>	void i2c_smbus_alert_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable SMBus Alert
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SMBus Alert */
i2c_smbus_alert_disable(I2C0);
```

## i2c\_smbus\_default\_addr\_enable

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

**Table 3-473. Function i2c\_smbus\_default\_addr\_enable**

<b>Function name</b>	i2c_smbus_default_addr_enable
<b>Function prototype</b>	void i2c_smbus_default_addr_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable SMBus device default address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SMBus device default address */
```

```
i2c_smbus_default_addr_enable(I2C0);
```

## i2c\_smbus\_default\_addr\_disable

The description of i2c\_smbus\_default\_addr\_disable is shown as below:

**Table 3-474. Function i2c\_smbus\_default\_addr\_disable**

<b>Function name</b>	i2c_smbus_default_addr_disable
<b>Function prototype</b>	void i2c_smbus_default_addr_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable SMBus device default address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SMBus device default address */
```

```
i2c_smbus_default_addr_disable(I2C0);
```

## i2c\_smbus\_host\_addr\_enable

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

**Table 3-475. Function i2c\_smbus\_host\_addr\_enable**

<b>Function name</b>	i2c_smbus_host_addr_enable
<b>Function prototype</b>	void i2c_smbus_host_addr_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable SMBus Host address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SMBus Host address */
i2c_smbus_host_addr_enable(I2C0);
```

## i2c\_smbus\_host\_addr\_disable

The description of i2c\_smbus\_host\_addr\_disable is shown as below:

**Table 3-476. Function i2c\_smbus\_host\_addr\_disable**

<b>Function name</b>	i2c_smbus_host_addr_disable
<b>Function prototype</b>	void i2c_smbus_host_addr_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable SMBus Host address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SMBus Host address */
i2c_smbus_host_addr_disable(I2C0);
```

## i2c\_extended\_clock\_timeout\_enable

The description of i2c\_extended\_clock\_timeout\_enable is shown as below:

**Table 3-477. Function i2c\_extended\_clock\_timeout\_enable**

<b>Function name</b>	i2c_extended_clock_timeout_enable
<b>Function prototype</b>	void i2c_extended_clock_timeout_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable extended clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable extended clock timeout detection */
i2c_extended_clock_timeout_enable(I2C0);
```

## i2c\_extended\_clock\_timeout\_disable

The description of i2c\_extended\_clock\_timeout\_disable is shown as below:

**Table 3-478. Function i2c\_extended\_clock\_timeout\_disable**

<b>Function name</b>	i2c_extended_clock_timeout_disable
<b>Function prototype</b>	void i2c_extended_clock_timeout_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable extended clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable extended clock timeout detection */
i2c_extended_clock_timeout_disable(I2C0);
```

## i2c\_clock\_timeout\_enable

The description of i2c\_clock\_timeout\_enable is shown as below:

**Table 3-479. Function i2c\_clock\_timeout\_enable**

<b>Function name</b>	i2c_clock_timeout_enable
<b>Function prototype</b>	void i2c_clock_timeout_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	enable clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable clock timeout detection */
i2c_clock_timeout_enable(I2C0);
```

## i2c\_clock\_timeout\_disable

The description of i2c\_clock\_timeout\_disable is shown as below:

**Table 3-480. Function i2c\_clock\_timeout\_disable**

<b>Function name</b>	i2c_clock_timeout_disable
<b>Function prototype</b>	void i2c_clock_timeout_disable(uint32_t i2c_periph);
<b>Function descriptions</b>	disable clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable clock timeout detection */
i2c_clock_timeout_disable(I2C0);
```



## i2c\_bus\_timeout\_b\_config

The description of i2c\_bus\_timeout\_b\_config is shown as below:

**Table 3-481. Function i2c\_bus\_timeout\_b\_config**

<b>Function name</b>	i2c_bus_timeout_b_config
<b>Function prototype</b>	void i2c_bus_timeout_b_config(uint32_t i2c_periph, uint32_t timeout);
<b>Function descriptions</b>	configure bus timeout B
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>timeout</b>	0x00000000-0x00000FFF, bus timeout B
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure bus timeout B */
```

```
i2c_bus_timeout_b_config(I2C0, 0xff);
```

## i2c\_bus\_timeout\_a\_config

The description of i2c\_bus\_timeout\_a\_config is shown as below:

**Table 3-482. Function i2c\_bus\_timeout\_a\_config**

<b>Function name</b>	i2c_bus_timeout_a_config
<b>Function prototype</b>	void i2c_bus_timeout_a_config(uint32_t i2c_periph, uint32_t timeout);
<b>Function descriptions</b>	configure bus timeout A
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>timeout</b>	0x00000000-0x00000FFF, bus timeout A
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure bus timeout A */
```

```
i2c_bus_timeout_a_config(I2C0, 0xff);
```

### i2c\_idle\_clock\_timeout\_config

The description of i2c\_idle\_clock\_timeout\_config is shown as below:

**Table 3-483. Function i2c\_idle\_clock\_timeout\_config**

<b>Function name</b>	i2c_idle_clock_timeout_config
<b>Function prototype</b>	void i2c_idle_clock_timeout_config(uint32_t i2c_periph, uint32_t timeout);
<b>Function descriptions</b>	configure idle clock timeout detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>timeout</b>	bus timeout A
<i>BUSTOA_DETECT_SCL_LOW</i>	BUSTOA is used to detect SCL low timeout
<i>BUSTOA_DETECT_IDLE</i>	BUSTOA is used to detect both SCL and SDA high timeout when the bus is idle
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure idle clock timeout detection */
```

```
i2c_idle_clock_timeout_config(I2C0, BUSTOA_DETECT_SCL_LOW);
```

### i2c\_flag\_get

The description of i2c\_flag\_get is shown as below:

**Table 3-484. Function i2c\_flag\_get**

<b>Function name</b>	i2c_flag_get
<b>Function prototype</b>	FlagStatus i2c_flag_get(uint32_t i2c_periph, uint32_t flag);
<b>Function descriptions</b>	get I2C flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)

Input parameter{in}	
<b>flag</b>	I2C flags
<i>I2C_FLAG_TBE</i>	I2C_TDATA is empty during transmitting
<i>I2C_FLAG_TI</i>	transmit interrupt
<i>I2C_FLAG_RBNE</i>	I2C_RDATA is not empty during receiving
<i>I2C_FLAG_ADDSEND</i>	address received matches in slave mode
<i>I2C_FLAG_NACK</i>	not acknowledge flag
<i>I2C_FLAG_STPDET</i>	STOP condition detected in slave mode
<i>I2C_FLAG_TC</i>	transfer complete in master mode
<i>I2C_FLAG_TCR</i>	transfer complete reload
<i>I2C_FLAG_BERR</i>	bus error
<i>I2C_FLAG_LOSTARB</i>	arbitration Lost
<i>I2C_FLAG_OUERR</i>	overrun/underrun error in slave mode
<i>I2C_FLAG_PECERR</i>	PEC error
<i>I2C_FLAG_TIMEOUT</i>	timeout flag
<i>I2C_FLAG_SMBALT</i>	SMBus Alert
<i>I2C_FLAG_I2CBSY</i>	busy flag
<i>I2C_FLAG_TR</i>	whether the I2C is a transmitter or a receiver in slave mode
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get I2C flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get(I2C0, I2C_FLAG_TBE);
```

### i2c\_flag\_clear

The description of i2c\_flag\_clear is shown as below:

**Table 3-485. 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>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
Input parameter{in}	
<b>flag</b>	I2C flags

<i>I2C_FLAG_ADDSEND</i>	address received matches in slave mode
<i>I2C_FLAG_NACK</i>	not acknowledge flag
<i>I2C_FLAG_STPDET</i>	STOP condition detected in slave mode
<i>I2C_FLAG_BERR</i>	bus error
<i>I2C_FLAG_LOSTARB</i>	arbitration Lost
<i>I2C_FLAG_OUERR</i>	overrun/underrun error in slave mode
<i>I2C_FLAG_PECERR</i>	PEC error
<i>I2C_FLAG_TIMEOUT</i>	timeout flag
<i>I2C_FLAG_SMBALT</i>	SMBus Alert
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear a bus error flag*/
```

```
i2c_flag_clear(I2C0, I2C_FLAG_BERR);
```

## i2c\_interrupt\_enable

The description of i2c\_interrupt\_enable is shown as below:

**Table 3-486. Function i2c\_interrupt\_enable**

<b>Function name</b>	i2c_interrupt_enable
<b>Function prototype</b>	void i2c_interrupt_enable(uint32_t i2c_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable I2C interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>interrupt</b>	I2C interrupts
<i>I2C_INT_ERR</i>	error interrupt
<i>I2C_INT_TC</i>	transfer complete interrupt
<i>I2C_INT_STPDET</i>	stop detection interrupt
<i>I2C_INT_NACK</i>	not acknowledge received interrupt
<i>I2C_INT_ADDM</i>	address match interrupt
<i>I2C_INT_RBNE</i>	receive interrupt
<i>I2C_INT_TI</i>	transmit interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* enable I2C0 transmit interrupt */
```

```
i2c_interrupt_enable(I2C0, I2C_INT_TI);
```

### i2c\_interrupt\_disable

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

**Table 3-487. Function i2c\_interrupt\_disable**

<b>Function name</b>	i2c_interrupt_disable
<b>Function prototype</b>	void i2c_interrupt_disable(uint32_t i2c_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable I2C interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>interrupt</b>	I2C interrupts
<i>I2C_INT_ERR</i>	error interrupt
<i>I2C_INT_TC</i>	transfer complete interrupt
<i>I2C_INT_STPDET</i>	stop detection interrupt
<i>I2C_INT_NACK</i>	not acknowledge received interrupt
<i>I2C_INT_ADDM</i>	address match interrupt
<i>I2C_INT_RBNE</i>	receive interrupt
<i>I2C_INT_TI</i>	transmit interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 transmit interrupt */
```

```
i2c_interrupt_disable(I2C0, I2C_INT_TI);
```

### i2c\_interrupt\_flag\_get

The description of i2c\_interrupt\_flag\_get is shown as below:

**Table 3-488. Function i2c\_interrupt\_flag\_get**

<b>Function name</b>	i2c_interrupt_flag_get
<b>Function prototype</b>	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph,

	i2c_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	get I2C interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>int_flag</b>	I2C interrupt flags, refer to <a href="#">Table 3-429. i2c_interrupt_flag_enum</a> .
<i>I2C_INT_FLAG_TI</i>	transmit interrupt flag
<i>I2C_INT_FLAG_RBNE</i>	I2C_RDATA is not empty during receiving interrupt flag
<i>I2C_INT_FLAG_ADDS END</i>	address received matches in slave mode interrupt flag
<i>I2C_INT_FLAG_NACK</i>	not acknowledge interrupt flag
<i>I2C_INT_FLAG_STPD ET</i>	stop condition detected in slave mode interrupt flag
<i>I2C_INT_FLAG_TC</i>	transfer complete in master mode interrupt flag
<i>I2C_INT_FLAG_TCR</i>	transfer complete reload interrupt flag
<i>I2C_INT_FLAG_BERR</i>	bus error interrupt flag
<i>I2C_INT_FLAG_LOSTA RB</i>	arbitration lost interrupt flag
<i>I2C_INT_FLAG_OUER R</i>	overflow/underrun error in slave mode interrupt flag
<i>I2C_INT_FLAG_PEC RR</i>	PEC error interrupt flag
<i>I2C_INT_FLAG_TIMEO UT</i>	timeout interrupt flag
<i>I2C_INT_FLAG_SMBA LT</i>	SMBus Alert interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* get I2C interrupt flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get(I2C0, I2C_INT_FLAG_TI);
```

### i2c\_interrupt\_flag\_clear

The description of i2c\_interrupt\_flag\_clear is shown as below:

Table 3-489. Function i2c\_interrupt\_flag\_clear

Function name	i2c_interrupt_flag_clear
Function prototype	void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
Function descriptions	clear I2C interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
Input parameter{in}	
int_flag	I2C interrupt flags, refer to <a href="#">Table 3-429. i2c_interrupt_flag_enum</a> .
I2C_INT_FLAG_ADDS END	address received matches in slave mode interrupt flag
I2C_INT_FLAG_NACK	not acknowledge interrupt flag
I2C_INT_FLAG_STPD ET	stop condition detected in slave mode interrupt flag
I2C_INT_FLAG_BERR	bus error interrupt flag
I2C_INT_FLAG_LOSTA RB	arbitration lost interrupt flag
I2C_INT_FLAG_OUER R	overflow/underrun error in slave mode interrupt flag
I2C_INT_FLAG_PECE RR	PEC error interrupt flag
I2C_INT_FLAG_TIMEO UT	timeout interrupt flag
I2C_INT_FLAG_SMBA LT	SMBus Alert interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear a bus error flag */
```

```
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_BERR);
```

## 3.19. MISC

MISC is a software package that provide the interfaces for NVIC and SysTick. The NVIC and SysTick registers are listed in chapter [3.19.1](#), the MISC firmware functions are introduced in chapter [3.19.2](#).

### 3.19.1. Descriptions of Peripheral registers

**Table 3-490. 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
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\_cm33.h file

2. refer to the structure SCB\_Type, is defined in the core\_cm33.h file



**Table 3-491. 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

### 3.19.2. Descriptions of Peripheral functions

#### Enum IRQn\_Type

**Table 3-492. 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
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_IRQn	TIMER0 break interrupts
TIMER0_UP_IRQn	TIMER0 update interrupts
TIMER0_TRG_CMT_IRQn	TIMER0 trigger and commutation interrupts
TIMER0_Channel_IRQn	TIMER0 channel capture compare interrupt
TIMER1_IRQn	TIMER1 interrupt

Member name	Function description
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_IRQn	TIMER7 break interrupts
TIMER7_UP_IRQn	TIMER7 update interrupts
TIMER7_TRG_CMT_IRQn	TIMER7 trigger and commutation interrupts
TIMER7_Channel_IRQn	TIMER7 channel capture compare interrupt
ADC2_IRQn	ADC2 global interrupt
RCU_CKFM_IRQn	RCU clock frequency monitor interrupt
CMP_WAKEUP_IRQn	CMP wakeup through EXTI line interrupt
SPI2_IRQn	SPI2 global interrupt
UART3_IRQn	UART3 global interrupt
UART4_IRQn	UART4 global interrupt
TIMER5_IRQn	TIMER5 global interrupt
TIMER6_IRQn	TIMER6 global interrupt
DMA1_Channel0_IRQn	DMA1 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
DAC_IRQn	DAC global interrupt
PMU_UVD_OVD_IRQn	VUVD/VOVD 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
SYSTEM_IRQn	System interrupt
FPU_IRQn	FPU interrupt
CMP_IRQn	CMP interrupt
DMAMUX_IRQn	DMAMUX interrupt
CAU_IRQn	CAU interrupt

Member name	Function description
HAU_IRQn	HAU interrupt
TRNG_IRQn	TRNG interrupt
USBFS_IRQn	USBFS interrupt
TIMER4_IRQn	TIMER4 interrupt
TIMER15_IRQn	TIMER15 interrupt
TIMER16_IRQn	TIMER16 interrupt
TIMER0_BRK_Channel_IRQn	TIMER4 break interrupt
TIMER7_BRK_Channel_IRQn	TIMER4 break interrupt

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

**Table 3-493. MISC firmware function**

Function name	Function description
<code>nvic_priority_group_set</code>	set the priority group
<code>nvic_irq_enable</code>	enable NVIC interrupt request
<code>nvic_irq_disable</code>	disable NVIC interrupt request
<code>nvic_system_reset</code>	request system reset
<code>nvic_vector_table_set</code>	set the NVIC vector table address
<code>system_lowpower_set</code>	set the state of the low power mode
<code>system_lowpower_reset</code>	reset the state of the low power mode
<code>systick_clksource_set</code>	set the systick clock source

## nvic\_priority\_group\_set

The description of `nvic_priority_group_set` is shown as below:

**Table 3-494. Function `nvic_priority_group_set`**

Function name	<code>nvic_priority_group_set</code>
Function prototype	<code>void nvic_priority_group_set(uint32_t nvic_prigroup);</code>
Function descriptions	set the priority group
Precondition	-
The called functions	-
Input parameter{in}	
<code>nvic_prigroup</code>	priority group
<code>NVIC_PRIGROUP_PR</code> <code>E0_SUB4</code>	0 bits for pre-emption priority 4 bits for subpriority
<code>NVIC_PRIGROUP_PR</code> <code>E1_SUB3</code>	1 bits for pre-emption priority 3 bits for subpriority
<code>NVIC_PRIGROUP_PR</code> <code>E2_SUB2</code>	2 bits for pre-emption priority 2 bits for subpriority
<code>NVIC_PRIGROUP_PR</code> <code>E3_SUB1</code>	3 bits for pre-emption priority 1 bits for subpriority
<code>NVIC_PRIGROUP_PR</code> <code>E4_SUB0</code>	4 bits for pre-emption priority 0 bits for subpriority

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* priority group configuration , 0 bits for pre-emption priority 4 bits for subpriority */
```

```
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

## nvic\_irq\_enable

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

**Table 3-495. Function nvic\_irq\_enable**

Function name	nvic_irq_enable
Function prototype	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
Function descriptions	enable NVIC request
Precondition	-
The called functions	nvic_priority_group_set
Input parameter{in}	
nvic_irq	NVIC interrupt, refer to enum <a href="#">Enum IRQn_Type</a>
Input parameter{in}	
nvic_irq_pre_priority	the pre-emption priority needed to set
Input parameter{in}	
nvic_irq_sub_priority	the subpriority needed to set
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable window watchDog timer interrupt , pre-emption priority is 1, subpriority is 1 */
```

```
nvic_irq_enable(WWDGT_IRQn, 1, 1);
```

## nvic\_irq\_disable

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

**Table 3-496. Function nvic\_irq\_disable**

Function name	nvic_irq_disable
Function prototype	void nvic_irq_disable(uint8_t nvic_irq);
Function descriptions	disable NVIC request
Precondition	-
The called functions	-

Input parameter{in}	
<b>nvic_irq</b>	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\_system\_reset**

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

**Table 3-497. Function nvic\_system\_reset**

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

Example:

```
/* request system reset */
nvic_system_reset ();
```

### **nvic\_vector\_table\_set**

The description of nvic\_vector\_table\_set is shown as below:

**Table 3-498. Function nvic\_vector\_table\_set**

<b>Function name</b>	nvic_vector_table_set
<b>Function prototype</b>	void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);
<b>Function descriptions</b>	set the NVIC vector table base address
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>nvic_vect_tab</b>	the RAM or FLASH base address
<i>NVIC_VECTTAB_RAM</i>	RAM base address

<code>NVIC_VECTTAB_FLASH</code> <i>H</i>	Flash base address
<b>Input parameter{in}</b>	
<b>offset</b>	vector table offset (vector table start address= base address+offset)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set vector table address = NVIC_VECTTAB_FLASH +0x200 */
nvic_vector_table_set (NVIC_VECTTAB_FLASH,0x200);
```

### system\_lowpower\_set

The description of system\_lowpower\_set is shown as below:

**Table 3-499. 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
<code>SCB_LPM_SLEEP_EXIT_ISR</code>	if chose this para, the system always enter low power mode by exiting from ISR
<code>SCB_LPM_DEEPSLEEP_P</code>	if chose this para, the system will enter the DEEPSLEEP mode
<code>SCB_LPM_WAKE_BY_ALL_INT</code>	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 3-500. Function `system_lowpower_reset`

<b>Function name</b>	<code>system_lowpower_reset</code>
<b>Function prototype</b>	<code>void system_lowpower_reset(uint8_t lowpower_mode);</code>
<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
<code>SCB_LPM_SLEEP_EXIT_ISR</code>	if chose this para, the system will exit low power mode by exiting from ISR
<code>SCB_LPM_DEEPSLEEP</code>	if chose this para, the system will enter the SLEEP mode
<code>SCB_LPM_WAKE_BY_ALL_INT</code>	if chose this para, the lowpower mode only can be woke up by the enable interrupts
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* the system will exit low power mode by exiting from ISR */
system_lowpower_reset (SCB_LPM_SLEEP_EXIT_ISR);
```

### **systick\_clksource\_set**

The description of `systick_clksource_set` is shown as below:

Table 3-501. Function `systick_clksource_set`

<b>Function name</b>	<code>systick_clksource_set</code>
<b>Function prototype</b>	<code>void systick_clksource_set(uint32_t systick_clksource);</code>
<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
<code>SYSTICK_CLKSOURC_E_HCLK</code>	systick clock source is from HCLK
<code>SYSTICK_CLKSOURC_E_HCLK_DIV8</code>	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);
```

## 3.20. PMU

According to the Power management unit (PMU), provides three types of power saving modes, including Sleep, Deep-sleep and Standby mode. The PMU registers are listed in chapter [3.20.1](#), the PMU firmware functions are introduced in chapter [3.20.2](#).

### 3.20.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

**Table 3-502. PMU Registers**

Registers	Descriptions
PMU_CTL0	Control register 0
PMU_CS	Control and status register
PMU_CTL1	Control register 1

### 3.20.2. Descriptions of Peripheral functions

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

**Table 3-503. PMU firmware function**

Function name	Function description
pmu_deinit	deinitialize the PMU
pmu_lock	Lock pmu register write
Pmu_unlock	Unlock pmu register write
pmu_lvd_select	select low voltage detector threshold
pmu_pdrvs_select	select power down reset detector threshold
pmu_ldo_output_select	select LDO output voltage
pmu_lvd_disable	disable PMU lvd
pmu_lvd_enable	enable PMU lvd
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



Function name	Function description
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
pmu_backup_write_disable	disable backup domain write
pmu_vavd_disable	disable vavd
pmu_vavd_enable	enable vavd
pmu_vovd_disable	disable vovd
pmu_vovd_enable	enable vovd
pmu_vuvd_disable	disable vuvd
pmu_vuvd_enable	enable vuvd
pmu_vavd_select	select VAVD threshold
pmu_vovd_select	select vovdvc threshold
pmu_vuvd_select	select vuvdvc threshold
pmu_vuvdo_dnf_select	select vuvdo_dnf
pmu_vovdo_dnf_select	select vovdo_dnf
pmu_flag_clear	clear flag bit
pmu_flag_get	get flag state

## pmu\_deinit

The description of pmu\_deinit is shown as below:

**Table 3-504. Function pmu\_deinit**

Function name	pmu_deinit
Function prototype	void pmu_deinit(void);
Function descriptions	deinitialize the PMU

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

Example:

```
/* reset PMU */
```

```
pmu_deinit ();
```

### pmu\_lock

The description of pmu\_highdriver\_mode\_enable is shown as below:

**Table 3-505. Function pmu\_lock**

<b>Function name</b>	pmu_lock
<b>Function prototype</b>	void pmu_lock(void)
<b>Function descriptions</b>	lock the pmu register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock the pmu register */
```

```
pmu_lock ();
```

## pmu\_unlock

The description of pmu\_highdriver\_mode\_enable is shown as below:

**Table 3-506. Function pmu\_unlock**

<b>Function name</b>	pmu_unlock
<b>Function prototype</b>	void pmu_unlock(void)
<b>Function descriptions</b>	unlock the pmu register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* unlock the pmu register */
```

```
pmu_unlock ();
```

## pmu\_lvd\_select

The description of pmu\_lvd\_select is shown as below:

**Table 3-507. Function pmu\_lvd\_select**

<b>Function name</b>	pmu_lvd_select
<b>Function prototype</b>	void pmu_lvd_select(uint32_t lvdn);
<b>Function descriptions</b>	select low voltage detector threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lvdn</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\_pdrvs\_select

The description of pmu\_pdrvs\_select is shown as below:

**Table 3-508. Function pmu\_pdrvs\_select**

<b>Function name</b>	pmu_pdrvs_select
<b>Function prototype</b>	void pmu_pdrvs_select(uint32_t pdrv1);
<b>Function descriptions</b>	select power down reset voltage detector threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lvdt_n</b>	voltage threshold value
<i>PMU_PDRVS_LEVEL0</i>	voltage threshold is 2.35V
<i>PMU_PDRVS_LEVEL1</i>	voltage threshold is 1.8V
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* select low voltage detector threshold as 2.35V */
```

```
pmu_pdrvs_select (PMU_PDRVS_LEVEL0);
```

### pmu\_ldo\_output\_select

The description of pmu\_ldo\_output\_select is shown as below:

**Table 3-509. 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>	PMU LDO output voltage select definitions
<i>PMU_LDOVS_1</i>	LDO output voltage 0.9V
<i>PMU_LDOVS_2</i>	LDO output voltage 0.95V
<i>PMU_LDOVS_3</i>	LDO output voltage 1.0V
<i>PMU_LDOVS_4</i>	LDO output voltage 1.05V
<i>PMU_LDOVS_5</i>	LDO output voltage 1.1V
<i>PMU_LDOVS_6</i>	LDO output voltage 1.15V
<i>PMU_LDOVS_7</i>	LDO output voltage 1.2V
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select output low voltage mode */
```

```
pmu_ldo_output_select (PMU_LDOVS_LOW);
```

## pmu\_lvd\_enable

The description of pmu\_lvd\_enable is shown as below:

**Table 3-510. Function pmu\_lvd\_enable**

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

Example:

```
/* enable PMU lvd */
pmu_lvd_enable();
```

## pmu\_lvd\_disable

The description of pmu\_lvd\_disable is shown as below:

**Table 3-511. Function pmu\_lvd\_disable**

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

Example:

```
/* disable PMU lvd */
```

```
pmu_lvd_disable ();
```

### pmu\_lowdriver\_mode\_enable

The description of pmu\_lowdriver\_mode\_enable is shown as below:

**Table 3-512. 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>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-513. Function pmu\_lowdriver\_mode\_disable**

<b>Function name</b>	pmu_lowdriver_mode_disable
<b>Function prototype</b>	void pmu_lowdriver_mode_disable (void);
<b>Function descriptions</b>	disable low-driver mode in deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

```
/* disable low-driver mode in deep-sleep mode */
```

```
pmu_lowdriver_mode_disable ();
```

### pmu\_lowpower\_driver\_config

The description of pmu\_lowpower\_driver\_config is shown as below:

**Table 3-514. 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>	-
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* normal driver when use low power LDO */
```

```
pmu_lowpower_driver_config (PMU_NORMALDR_LOWPWR);
```



## pmu\_normalpower\_driver\_config

The description of pmu\_normalpower\_driver\_config is shown as below:

**Table 3-515. 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_NORMALPWR</i>	normal driver when use normal power LDO
<i>PMU_LOWDR_NORMALALPWR</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 3-516. 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>	-

Input parameter{in}	
<b>sleepmodecmd</b>	command to enter sleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PMU work at sleep mode */
```

```
pmu_to_sleepmode (WFI_CMD);
```

### pmu\_to\_deepsleepmode

The description of pmu\_to\_deepsleepmode is shown as below:

**Table 3-517. 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>	-
Input parameter{in}	
<b>ldo</b>	ldo work mode
<i>PMU_LDO_NORMAL</i>	LDO normal work when pmu enter deepsleep mode
<i>PMU_LDO_LOWPOWER</i>	LDO work at low power mode when pmu enter deepsleep mode
Input parameter{in}	
<i>PMU_LOWDRIVER_ENABLE</i>	low-driver mode enable in deep-sleep mode
<i>PMU_LOWDRIVER_DISABLE</i>	low-driver mode disable in deep-sleep mode

Input parameter{in}	
<b>deepsleepmodecmd</b>	command to enter deepsleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PMU work at deepsleep mode */
```

```
pmu_to_deepsleepmode (PMU_LDO_NORMAL, WFI_CMD);
```

### pmu\_to\_standbymode

The description of pmu\_to\_standbymode is shown as below:

**Table 3-518. Function pmu\_to\_standbymode**

<b>Function name</b>	pmu_to_standbymode
<b>Function prototype</b>	void pmu_to_standbymode(void);
<b>Function descriptions</b>	pmu work at standby mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PMU work at standby mode */
```

```
pmu_to_standby ();
```

## pmu\_wakeup\_pin\_enable

The description of pmu\_wakeup\_pin\_enable is shown as below:

**Table 3-519. 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 3-520. 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\_backup\_write\_enable

The description of pmu\_backup\_write\_enable is shown as below:

**Table 3-521. Function pmu\_backup\_write\_enable**

Function name	pmu_backup_write_enable
Function prototype	void pmu_backup_write_enable (void);
Function descriptions	enable backup domain write
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable backup domain write */
```

```
pmu_backup_write_enable ();
```

### pmu\_backup\_write\_disable

The description of pmu\_backup\_write\_disable is shown as below:

**Table 3-522. Function pmu\_backup\_write\_disable**

Function name	pmu_backup_write_disable
Function prototype	void pmu_backup_write_disable (void);
Function descriptions	disable backup domain write

<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\_vavd\_enable

The description of pmu\_vavd\_enable is shown as below:

**Table 3-523. Function pmu\_vavd\_enable**

<b>Function name</b>	pmu_vavd_enable
<b>Function prototype</b>	void pmu_vavd_enable(void);
<b>Function descriptions</b>	enable PMU analog voltage detector
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable PMU analog voltage detector */
```

```
pmu_vavd_enable();
```

### pmu\_vavd\_disable

The description of pmu\_vavd\_disable is shown as below:

**Table 3-524. Function pmu\_vavd\_disable**

<b>Function name</b>	pmu_vavd_disable
----------------------	------------------

<b>Function prototype</b>	void pmu_vavd_disable(void);
<b>Function descriptions</b>	disable PMU analog voltage detector
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable PMU analog voltage detector */
pmu_vavd_disable();
```

### pmu\_vovd\_enable

The description of pmu\_vovd\_enable is shown as below:

**Table 3-525. Function pmu\_vovd\_enable**

<b>Function name</b>	pmu_vovd_enable
<b>Function prototype</b>	void pmu_vovd_enable(void);
<b>Function descriptions</b>	enable PMU over voltage detector
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable PMU voltage detector */
pmu_vovd_enable();
```

### pmu\_vovd\_disable

The description of pmu\_vovd\_disable is shown as below:

**Table 3-526. Function pmu\_vovd\_disable**

<b>Function name</b>	pmu_vovd_disable
<b>Function prototype</b>	void pmu_vovd_disable(void);
<b>Function descriptions</b>	disable PMU over voltage detector

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

Example:

```
/* disable PMUvoltage detector */
```

```
pmu_vovd_disable();
```

### pmu\_vuvd\_enable

The description of pmu\_vuvd\_enable is shown as below:

**Table 3-527. Function pmu\_vuvd\_enable**

Function name	pmu_vuvd_enable
Function prototype	void pmu_vuvd_enable (void);
Function descriptions	enable PMU core under voltage detector
Precondition	-
The called functions	-
Input parameter{in}	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable PMU core under voltage detector */
```

```
pmu_vuvd_enable();
```

### pmu\_vuvd\_disable

The description of pmu\_vuvd\_disable is shown as below:

**Table 3-528. Function pmu\_vuvd\_disable**

Function name	pmu_vuvd_disable
Function prototype	void pmu_vuvd_disable (void);
Function descriptions	disable PMU core under voltage detector
Precondition	-
The called functions	-



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

Example:

```
/* disable PMU core under voltage detector */
```

```
pmu_vuvd_disable();
```

### pmu\_vavd\_select

The description of pmu\_vavd\_select is shown as below:

**Table 3-529. Function pmu\_avd\_select**

Function name	pmu_vavd_select
Function prototype	void pmu_vavd_select(uint32_t avdt_n);
Function descriptions	select analog voltage detector threshold
Precondition	-
The called functions	-
Input parameter{in}	
avdt_n	select analog voltage detector threshold
PMU_VAVDVC_0	voltage threshold of analog voltage detector is 2.3V
PMU_VAVDVC_1	voltage threshold of analog voltage detector is 2.5V
PMU_VAVDVC_2	voltage threshold of analog voltage detector is 2.7V
PMU_VAVDVC_3	voltage threshold of analog voltage detector is 2.9V
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select analog voltage detector threshold 2.9V */
```

```
pmu_vavd_select(PMU_VAVDVC_3);
```

### pmu\_vovd\_select

The description of pmu\_vovd\_select is shown as below:

**Table 3-530. Function pmu\_vovd\_enable**

Function name	pmu_vovd_select
Function prototype	void pmu_vovd_select (uint32_t ovdt_n);
Function descriptions	enable PMU voltage detector

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
ovdt_n	voltage threshold value
PMU_VOVDVC_0	voltage threshold is 1.25V
PMU_VOVDVC_1	voltage threshold is 1.30V
PMU_VOVDVC_2	voltage threshold is 1.35V
PMU_VOVDVC_3	voltage threshold is 1.40V
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select PMU core over voltage detector threshold 1.25V */
```

```
pmu_vovd_select (PMU_VOVDVC_0);
```

### pmu\_vuvd\_select

The description of pmu\_vuvd\_select is shown as below:

**Table 3-531. Function pmu\_vuvd\_select**

<b>Function name</b>	pmu_vuvd_select
<b>Function prototype</b>	void pmu_vuvd_select (uint32_t ldo_n);
<b>Function descriptions</b>	select PMU core under voltage detector threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
ldo_n	Select LDO output voltage
PMU_VUVDVC_0	LDO output voltage 1.05V mode
PMU_VUVDVC_1	LDO output voltage 0.95V mode
PMU_VUVDVC_2	LDO output voltage 0.85V mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select LDO output voltage 0.85V */
```

```
pmu_vuvd_select(PMU_VUVDVC_2);
```

### pmu\_vovdo\_dnf\_select

The description of pmu\_vovdo\_dnf\_select is shown as below:

**Table 3-532. Function pmu\_vovdo\_dnf\_enable**

<b>Function name</b>	pmu_vovdo_dnf_select
<b>Function prototype</b>	void pmu_vovdo_dnf_select (uint32_t vovdo_value);
<b>Function descriptions</b>	Select length of digital filtering for over voltage
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
vovdo_value	0x00-0xff
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* Filtering peak length*/
pmu_vovdo_dnf_select (0x55);
```

### pmu\_vuvdo\_dnf\_select

The description of pmu\_vuvdo\_dnf\_select is shown as below:

**Table 3-533. Function pmu\_vuvdo\_dnf\_select**

<b>Function name</b>	pmu_vuvdo_dnf_select
<b>Function prototype</b>	void pmu_vuvdo_dnf_select (uint32_t vuvdo_value);
<b>Function descriptions</b>	select PMU core under voltage detector threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
vuvdo_value	0x00-0xff
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* Filtering peak length */
pmu_vuvdo_dnf_select(0x55);
```

### pmu\_flag\_get

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

**Table 3-534. 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_VAVDF</i>	LDO voltage select ready flag
<i>PMU_FLAG_VUVDFO</i>	VCORE low voltage detector flag 0
<i>PMU_FLAG_VOVDFO</i>	VCORE over voltage detector flag
<i>PMU_FLAG_VUVDFO1</i>	VCORE low voltage detector flag 1
<i>PMU_FLAG_LDOVSRF</i>	LDO voltage select ready flag
<i>PMU_FLAG_LDRF</i>	low-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 3-535. 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
<i>PMU_FLAG_RESET_WAKEUP</i>	reset wakeup flag
<i>PMU_FLAG_RESET_S TANDBY</i>	reset standby flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear flag bit */
```

```
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

## 3.21. RCU

RCU is the reset and clock unit. Reset Control includes the control of three kinds of reset: power reset, system reset and backup domain reset. The Clock Control unit provides a range of frequencies and clock functions. The RCU registers are listed in chapter [3.21.1](#), the RCU firmware functions are introduced in chapter [3.21.2](#).

### 3.21.1. Descriptions of Peripheral registers

RCU registers are listed in the table shown as below:

**Table 3-536. 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	AHB1 enable register

RCU_APB2EN	APB2 enable register
RCU_APB1EN	APB1 enable register
RCU_BDCTL	backup domain control register
RCU_RSTSCK	reset source / clock register
RCU_AHBRST	AHB reset register
RCU_CFG1	clock configuration register 1
RCU_PLLBWCFG	PLL bandwidth configuration register
RCU_DSV	deep-sleep mode voltage register
RCU_CKFMCFG0	clock frequency monitor configuration register 0
RCU_CKFMCFG1	clock frequency monitor configuration register 1
RCU_CKFMCFG2	clock frequency monitor configuration register 2
RCU_CKFMCFG3	clock frequency monitor configuration register 3
RCU_ADDCTL	Additional clock control register
RCU_ADDINT	Additional clock interrupt register
RCU_ADDAPB1RST	APB1 additional reset register
RCU_ADDAPB1EN	APB1 additional enable register
RCU_LOCK	lock register

### 3.21.2. Descriptions of Peripheral functions

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

**Table 3-537. RCU firmware function**

Function name	Function description
rcu_deinit	deinitialize the RCU
rcu_periph_clock_enable	enable the peripherals clock
rcu_periph_clock_disable	disable the peripherals clock
rcu_periph_clock_sleep_enable	enable the peripherals clock when sleep mode
rcu_periph_clock_sleep_disable	disable the peripherals clock when sleep mode
rcu_periph_reset_enable	reset the peripherals
rcu_periph_reset_disable	disable reset the peripherals
rcu_bkp_reset_enable	reset the BKP domain
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_ckout_config	configure the CK_OUT clock source
rcu_pll0_config	configure the PLL0 clock
rcu_pll1_config	configure the PLL1 clock
rcu_prediv0_config	configure the PREDIV0 division factor
rcu_prediv1_config	configure the PREDIV1 division factor
rcu_adc_clock_config	configure the ADC prescaler factor
rcu_usb_clock_config	configure the USBFS 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 source selection
rcu_fmc_clock_config	configure the FMC clock source selection
rcu_lxtal_drive_capability_config	configure the LXTAL drive capability
rcu_osci_stab_wait	wait for oscillator stabilization flags is SET or oscillator startup is timeout
rcu_osci_on	turn on the oscillator
rcu_osci_off	turn off the oscillator
rcu_osci_bypass_mode_enable	enable the oscillator bypass mode, HXTALEN or LXTALEN must be reset before it
rcu_osci_bypass_mode_disable	disable the oscillator bypass mode, HXTALEN or LXTALEN must be reset before it
rcu_irc8m_adjust_value_set	set the IRC8M adjust value
rcu_hxtal_clock_monitor_enable	enable the HXTAL clock monitor
rcu_hxtal_clock_monitor_disable	disable the HXTAL clock monitor
rcu_lxtal_clock_monitor_enable	enable the LXTAL clock monitor
rcu_lxtal_clock_monitor_disable	disable the LXTAL clock monitor
rcu_clock_freq_monitor_enable	enable the clock frequency monitor
rcu_clock_freq_monitor_disable	disable the clock frequency monitor
rcu_irc8m_freq_monitor_config	IRC8M clock frequency monitor range configuration
rcu_hxtal_monitor_threshold_config	HXTAL clock frequency monitor threshold configuration

rcu_pll0p_monitor_thres hold_config	PLL0P clock frequency monitor threshold configuration
rcu_pll1_monitor_thres old_config	PLL1 clock frequency monitor threshold configuration
rcu_deepsleep_voltage _set	set the deep sleep mode voltage
rcu_deepsleep_switch_ delay_set	delay before switch to IRC8M clock and enter deep-sleep mode
rcu_reg_lock	lock rcu register
rcu_reg_unlock	unlock rcu register
rcu_pll_bandwidth_confi g	config PLL0/PLL1 bandwidth
rcu_clock_freq_get	get the system clock, bus and peripheral clock frequency
rcu_flag_get	get the clock stabilization, peripheral reset and clock frequency failure flags
rcu_flag_clear	clear clock frequency failure flags
rcu_all_reset_flag_clear	clear all the reset flag
rcu_interrupt_enable	enable the clock interrupt
rcu_interrupt_disable	disable the clock interrupt
rcu_interrupt_flag_get	get the clock interrupt flags
rcu_interrupt_flag_clear	clear the clock interrupt flags

## Enum rcu\_periph\_enum

**Table 3-538. Enum rcu\_periph\_enum**

Member name	Descriptions
RCU_DMA0	DMA0 clock
RCU_DMA1	DMA1 clock
RCU_CRC	CRC clock
RCU_EXMC	EXMC clock
RCU_USBFS	USBFS clock
RCU_CAU	CAU clock
RCU_HAU	HAU clock
RCU_TRNG	TRNG clock
RCU_DMAMUX	DMAMUX clock
RCU_TIMER1	TIMER1 clock
RCU_TIMER2	TIMER2 clock
RCU_TIMER3	TIMER3 clock
RCU_TIMER4	TIMER4 clock
RCU_TIMER5	TIMER5 clock
RCU_TIMER6	TIMER6 clock
RCU_TIMER16	TIMER16 clock
RCU_WWDGT	WWDGT clock



RCU_SPI1	SPI1 clock
RCU_SPI2	SPI2 clock
RCU_USART1	USART1 clock
RCU_USART2	USART2 clock
RCU_UART3	UART3 clock
RCU_UART4	UART4 clock
RCU_I2C0	I2C0 clock
RCU_I2C1	I2C1 clock
RCU_CMP	CMP 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_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_TIMER15	TIMER16 clock
RCU_TRIGSEL	TRIGSEL clock
RCU_SYSCFG	SYSCFG clock
RCU_CAN0	CAN0 clock
RCU_CAN1	CAN1 clock

### Enum rcu\_periph\_sleep\_enum

Table 3-539. Enum rcu\_periph\_sleep\_enum

Member name	Descriptions
RCU_SRAM_SLP	SRAM interface clock enable when sleep mode
RCU_FMC_SLP	FMC clock enable when sleep mode

### Enum rcu\_periph\_reset\_enum

Table 3-540. Enum rcu\_periph\_reset\_enum

Member name	Descriptions
-------------	--------------

RCU_USBFSRST	USBFS clock reset
RCU_CAURST	CAU clock reset
RCU_HAURST	HAU clock reset
RCU_TRNGRST	TRNG clock reset
RCU_DMAMUXRST	DMAMUX clock reset
RCU_DMA0RST	DMA0X clock reset
RCU_DMA1RST	DMA1 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_TIMER16RST	TIMER16 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_UART4RST	UART4 clock reset
RCU_I2C0RST	I2C0 clock reset
RCU_I2C1RST	I2C1 clock reset
RCU_CMPRST	CMP 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_ADC0RST	ADC0 clock reset
RCU_ADC1RST	ADC1 clock reset
RCU_TIMER0RST	TIMER0 clock reset
RCU_SPI0RST	SPI0 clock reset
RCU_TIMER7RST	TIMER7 clock reset
RCU_USART0RST	USART0 clock reset
RCU_ADC2RST	ADC2 clock reset
RCU_TIMER15RST	TIMER16 clock reset
RCU_TRIGSELRST	TRIGSEL clock reset

RCU_SYSCFGRST	SYSCFG clock reset
RCU_CAN0RST	CAN0 clock reset
RCU_CAN1RST	CAN1 clock reset

## Enum rcu\_flag\_enum

**Table 3-541. Enum rcu\_flag\_enum**

Member name	Descriptions
RCU_FLAG_IRC8MST B	IRC8M stabilization flag
RCU_FLAG_HXTALST B	HXTAL stabilization flag
RCU_FLAG_PLL0STB	PLL0 stabilization flag
RCU_FLAG_PLL1STB	PLL1 stabilization flag
RCU_FLAG_LXTALST B	LXTAL stabilization flag
RCU_FLAG_IRC40KST B	IRC40K stabilization flag
RCU_FLAG_IRC48MS TB	IRC48M stabilization flag
RCU_FLAG_EPRST	external PIN reset flag
RCU_FLAG_PORRST	power reset flag
RCU_FLAG_SWRST	software reset flag
RCU_FLAG_FWDGTR ST	FWDGT reset flag
RCU_FLAG_WWDGTR ST	WWDGT reset flag
RCU_FLAG_LPRST	low-power reset flag
RCU_FLAG_IRC8MCK FF	IRC8M clock frequency failure flag
RCU_FLAG_HXTALCK FF	HXTAL clock frequency failure flag
RCU_FLAG_PLL0PCK FF	PLL0P clock frequency failure flag
RCU_FLAG_PLL1CKF F	PLL1 clock frequency failure flag

## Enum rcu\_int\_flag\_enum

**Table 3-542. Enum rcu\_int\_flag\_enum**

Member name	Descriptions
RCU_INT_FLAG_IRC4 0KSTB	IRC40K stabilization interrupt flag

RCU_INT_FLAG_LXTA LSTB	LXTAL stabilization interrupt flag
RCU_INT_FLAG_IRC8 MSTB	IRC8M stabilization interrupt flag
RCU_INT_FLAG_HXTA LSTB	HXTAL stabilization interrupt flag
RCU_INT_FLAG_PLL0 STB	PLL0 stabilization interrupt flag
RCU_INT_FLAG_PLL1 STB	PLL1 stabilization interrupt flag
RCU_INT_FLAG_HCK M	HXTAL clock stuck interrupt flag
RCU_INT_FLAG_LCK M	LXTAL clock stuck interrupt flag
RCU_INT_FLAG_IRC4 8MSTB	IRC48M stabilization interrupt flag

### Enum rcu\_flag\_clear\_enum

**Table 3-543. Enum rcu\_flag\_clear\_enum**

Member name	Descriptions
RCU_FLAG_IRC8MCK FF_CLR	IRC8M clock frequency failure interrupt flag clear
RCU_FLAG_HXTALCK FF_CLR	HXTAL clock frequency failure interrupt flag clear
RCU_FLAG_PLL0PCK FF_CLR	PLL0P clock frequency failure interrupt flag clear
RCU_FLAG_PLL1CKF F_CLR	PLL1 clock frequency failure interrupt flag clear

### Enum rcu\_int\_flag\_clear\_enum

**Table 3-544. Enum rcu\_int\_flag\_clear\_enum**

Member name	Descriptions
RCU_INT_FLAG_IRC4 0KSTB_CLR	IRC40K stabilization interrupt flags clear
RCU_INT_FLAG_LXTA LSTB_CLR	LXTAL stabilization interrupt flags clear
RCU_INT_FLAG_IRC8 MSTB_CLR	IRC8M stabilization interrupt flags clear
RCU_INT_FLAG_HXTA LSTB_CLR	HXTAL stabilization interrupt flags clear
RCU_INT_FLAG_PLL0 STB_CLR	PLL0 stabilization interrupt flags clear

RCU_INT_FLAG_PLL1 STB_CLR	PLL1 stabilization interrupt flags clear
RCU_INT_FLAG_HCK M_CLR	HXTAL clock stuck interrupt flags clear
RCU_INT_FLAG_LCK M_CLR	LXTAL clock stuck interrupt flags clear
RCU_INT_FLAG_IRC4 8MSTB_CLR	internal 48 MHz RC oscillator stabilization interrupt clear

### Enum rcu\_int\_enum

**Table 3-545. Enum rcu\_int\_enum**

Member name	Descriptions
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_PLL0STB	PLL0 stabilization interrupt
RCU_INT_PLL1STB	PLL1 stabilization interrupt
RCU_INT_LXTALCSS	LXTAL css interrupt
RCU_INT_IRC48MSTB	internal 48 MHz RC oscillator stabilization interrupt
RCU_INT_IRC8MCKFF	IRC8M clock frequency failure interrupt
RCU_INT_HXTALCKFF	HXTAL clock frequency failure interrupt
RCU_INT_PLL0PCKFF	PLL0P clock frequency failure interrupt
RCU_INT_PLL1CKFF	PLL1 clock frequency failure interrupt

### Enum rcu\_osci\_type\_enum

**Table 3-546. Enum rcu\_osci\_type\_enum**

Member name	Descriptions
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL
RCU_IRC8M	IRC8M
RCU_IRC48M	IRC48M
RCU_IRC40K	IRC40K
RCU_PLL0_CK	PLL0
RCU_PLL1_CK	PLL1

### Enum rcu\_clock\_freq\_enum

**Table 3-547. Enum rcu\_clock\_freq\_enum**

Member name	Descriptions
CK_SYS	system clock
CK_AHB	AHB clock

CK_APB1	APB1 clock
CK_APB2	APB2 clock
CK_PLL0	PLL0 clock

## Enum rcu\_ckfm\_enum

**Table 3-548. Enum rcu\_ckfm\_enum**

Member name	Descriptions
RCU_IRC8MCKFM	IRC8M clock frequency monitor
RCU_HXTALCKFM	HXTAL clock frequency monitor
RCU_PLL0PCKFM	PLL0P clock frequency monitor
RCU_PLL1CKFM	PLL1 clock frequency monitor

## rcu\_deinit

The description of rcu\_deinit is shown as below:

**Table 3-549. Function rcu\_deinit**

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

Example:

```
/* deinitialize the RCU */
```

```
rcu_deinit();
```

## rcu\_periph\_clock\_enable

The description of rcu\_periph\_clock\_enable is shown as below:

**Table 3-550. Function rcu\_periph\_clock\_enable**

Function name	rcu_periph_clock_enable
Function prototype	void rcu_periph_clock_enable(rcu_periph_enum periph)
Function descriptions	enable the peripherals clock
Precondition	-
The called functions	-
Input parameter{in}	

periph	RCU peripherals, refer to rcu_periph_enum
<i>RCU_DMAx</i> ( <i>x</i> = 0,1)	DMA clock
<i>RCU_CRC</i>	CRC clock
<i>RCU_EXMC</i>	EXMC clock
<i>RCU_USBFS</i>	USBFS clock
<i>RCU_CAU</i>	CAU clock
<i>RCU_HAU</i>	HAU clock
<i>RCU_TRNG</i>	TRNG clock
<i>RCU_DMAMUX</i>	DMAMUX clock
<i>RCU_TIMERx</i> ( <i>x</i> = 0,1,2,3,4,5,6,7,16)	TIMER clock
<i>RCU_WWDGT</i>	WWDGT clock
<i>RCU_SPIx</i> ( <i>x</i> = 0,1,2)	SPI clock
<i>RCU_USARTx</i> ( <i>x</i> = 0,1,2)	USART clock
<i>RCU_UARTx</i> ( <i>x</i> = 3,4)	UART clock
<i>RCU_I2Cx</i> ( <i>x</i> = 0,1)	I2C clock
<i>RCU_CMP</i>	CMP clock
<i>RCU_BKPI</i>	BKP interface clock
<i>RCU_PMU</i>	PMU clock
<i>RCU_DAC</i>	DAC clock
<i>RCU_RTC</i>	RTC clock
<i>RCU_CTC</i>	CTC clock
<i>RCU_AF</i>	alternate function clock
<i>RCU_GPIOx</i> ( <i>x</i> = A,B,C,D,E)	GPIO ports clock
<i>RCU_ADCx</i> ( <i>x</i> = 0,1,2)	ADC clock
<i>RCU_TRIGSEL</i>	TRIGSEL clock
<i>RCU_SYSCFG</i>	SYSCFG clock
<i>RCU_CANx</i> ( <i>x</i> = 0,1)	CAN clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the USART0 clock */
```

```
rcu_periph_clock_enable(RCU_USART0);
```

### rcu\_periph\_clock\_disable

The description of rcu\_periph\_clock\_disable is shown as below:

Table 3-551. 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 rcu_periph_enum
<i>RCU_DMAx</i> ( <i>x</i> = 0,1)	DMA clock
<i>RCU_CRC</i>	CRC clock
<i>RCU_EXMC</i>	EXMC clock
<i>RCU_USBFS</i>	USBFS clock
<i>RCU_CAU</i>	CAU clock
<i>RCU_HAU</i>	HAU clock
<i>RCU_TRNG</i>	TRNG clock
<i>RCU_DMAMUX</i>	DMAMUX clock
<i>RCU_TIMERx</i> ( <i>x</i> = 0,1,2,3,4,5,6,7,16)	TIMER clock
<i>RCU_WWDGT</i>	WWDGT clock
<i>RCU_SPIx</i> ( <i>x</i> = 0,1,2)	SPI clock
<i>RCU_USARTx</i> ( <i>x</i> = 0,1,2)	USART clock
<i>RCU_UARTx</i> ( <i>x</i> = 3,4)	UART clock
<i>RCU_I2Cx</i> ( <i>x</i> = 0,1)	I2C clock
<i>RCU_CMP</i>	CMP clock
<i>RCU_BKPI</i>	BKP interface clock
<i>RCU_PMU</i>	PMU clock
<i>RCU_DAC</i>	DAC clock
<i>RCU_RTC</i>	RTC clock
<i>RCU_CTC</i>	CTC clock
<i>RCU_AF</i>	alternate function clock
<i>RCU_GPIOx</i> ( <i>x</i> = A,B,C,D,E)	GPIO ports clock
<i>RCU_ADCx</i> ( <i>x</i> = 0,1,2)	ADC clock
<i>RCU_TRIGSEL</i>	TRIGSEL clock
<i>RCU_SYSCFG</i>	SYSCFG clock
<i>RCU_CANx</i> ( <i>x</i> = 0,1)	CAN clock
<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 3-552. 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 sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to rcu_periph_sleep_enum
<i>RCU_SRAM_SLP</i>	SRAM clock
<i>RCU_FMC_SLP</i>	FMC 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 3-553. 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 sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to rcu_periph_sleep_enum
<i>RCU_SRAM_SLP</i>	SRAM clock
<i>RCU_FMC_SLP</i>	FMC clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-554. Function rcu\_periph\_reset\_enable**

Function name	rcu_periph_reset_enable
Function prototype	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset)
Function descriptions	reset the peripherals
Precondition	-
The called functions	-
Input parameter{in}	
periph_reset	RCU peripherals reset, refer to rcu_periph_reset_enum
RCU_USBFSRST	reset USBFS
RCU_CAURST	reset CAU
RCU_HAURST	reset HAU
RCU_TRNGRST	reset TRNG
RCU_DMAMUXRST	reset DMAMUX
RCU_DMAxRST (x = 0,1)	DMA clock
RCU_TIMERxRST (x = 0,1,2,3,4,5,6,7,16)	reset TIMER
RCU_WWDGTRST	reset WWDGT
RCU_SPIxRST (x = 0,1,2)	reset SPI
RCU_USARTxRST (x = 0,1,2)	reset USART
RCU_UARTxRST (x = 3,4)	reset UART
RCU_I2CxRST (x = 0,1)	reset I2C
RCU_CMPRST	reset CMP
RCU_BKPIRST	reset BKPI
RCU_PMURST	reset PMU
RCU_DACRST	reset DAC
RCU_CTCRST	reset CTC
RCU_AFRST	reset alternate function clock
RCU_ADCRST (x = 0,1,2)	reset ADC
RCU_GPIOxRST (x = A,B,C,D,E)	reset GPIO ports

<i>RCU_TRIGSELRST</i>	reset TRIGSEL
<i>RCU_SYSCFGRST</i>	reset SYSCFG
<i>RCU_CANxRST</i> ( $x = 0, 1$ )	reset CAN
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-555. 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 reset the peripherals
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph_reset</b>	RCU peripherals reset, refer to rcu_periph_reset_enum
<i>RCU_USBFSRST</i>	reset USBFS
<i>RCU_CAURST</i>	reset CAU
<i>RCU_HAURST</i>	reset HAU
<i>RCU_TRNGRST</i>	reset TRNG
<i>RCU_DMAMUXRST</i>	reset DMAMUX
<i>RCU_DMAxRST</i> ( $x = 0, 1$ )	DMA clock
<i>RCU_TIMERxRST</i> ( $x = 0, 1, 2, 3, 4, 5, 6, 7, 16$ )	reset TIMER
<i>RCU_WWDGTRST</i>	reset WWDGT
<i>RCU_SPIxRST</i> ( $x = 0, 1, 2$ )	reset SPI
<i>RCU_USARTxRST</i> ( $x = 0, 1, 2$ )	reset USART
<i>RCU_UARTxRST</i> ( $x = 3, 4$ )	reset UART
<i>RCU_I2CxRST</i> ( $x = 0, 1$ )	reset I2C
<i>RCU_CMPRST</i>	reset CMP
<i>RCU_BKPIRST</i>	reset BKPI

<i>RCU_PMURST</i>	reset PMU
<i>RCU_DACRST</i>	reset DAC
<i>RCU_CTCRST</i>	reset CTC
<i>RCU_AFRST</i>	reset alternate function clock
<i>RCU_ADCRST</i> ( <i>x</i> = 0,1,2)	reset ADC
<i>RCU_GPIOxRST</i> ( <i>x</i> = A,B,C,D,E)	reset GPIO ports
<i>RCU_TRIGSELRST</i>	reset TRIGSEL
<i>RCU_SYSCFGRST</i>	reset SYSCFG
<i>RCU_CANxRST</i> ( <i>x</i> = 0,1)	reset CAN
<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 3-556. 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>	reset the BKP domain
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset the BKP domain */
rcu_bkp_reset_enable();
```

## rcu\_bkp\_reset\_disable

The description of rcu\_bkp\_reset\_disable is shown as below:

**Table 3-557. 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>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-558. 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_PLL0P</i>	select CK_PLL0P 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_CKSYS_SRC_HXTAL);
```

## rcu\_system\_clock\_source\_get

The description of rcu\_system\_clock\_source\_get is shown as below:

**Table 3-559. Function rcu\_system\_clock\_source\_get**

<b>Function name</b>	rcu_system_clock_source_get
<b>Function prototype</b>	uint32_t rcu_system_clock_source_get(void)
<b>Function descriptions</b>	get the system clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	which clock is selected as CK_SYS source
<i>RCU_SCSS_IRC8M</i>	CK_IRC8M is selected as the CK_SYS source
<i>RCU_SCSS_HXTAL</i>	CK_HXTAL is selected as the CK_SYS source
<i>RCU_SCSS_PLL0P</i>	CK_PLL0P is selected as the CK_SYS source

Example:

```
uint32_t temp_cksys_status;

/* get the CK_SYS source */

temp_cksys_status = rcu_system_clock_source_get();
```

## rcu\_ahb\_clock\_config

The description of rcu\_ahb\_clock\_config is shown as below:

**Table 3-560. Function rcu\_ahb\_clock\_config**

<b>Function name</b>	rcu_ahb_clock_config
<b>Function prototype</b>	void rcu_ahb_clock_config(uint32_t ck_ahb)
<b>Function descriptions</b>	configure the AHB clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_ahb</b>	AHB clock prescaler selection
<i>RCU_AHB_CKSYS_DIVx</i>	select CK_SYS / x as CK_AHB, (x = 1, 2, 4, 8, 16, 64, 128, 256, 512)
<b>Output parameter{out}</b>	
-	-

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 3-561. Function rcu\_apb1\_clock\_config**

Function name	rcu_apb1_clock_config
Function prototype	void rcu_apb1_clock_config(uint32_t ck_apb1)
Function descriptions	configure the APB1 clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CK_AHB/16 as CK_APB1 */
```

```
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);
```

### rcu\_apb2\_clock\_config

The description of rcu\_apb2\_clock\_config is shown as below:

**Table 3-562. Function rcu\_apb2\_clock\_config**

Function name	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_D</i> <i>IV1</i>	select CK_AHB as CK_APB2
<i>RCU_APB2_CKAHB_D</i> <i>IV2</i>	select CK_AHB / 2 as CK_APB2
<i>RCU_APB2_CKAHB_D</i> <i>IV4</i>	select CK_AHB / 4 as CK_APB2
<i>RCU_APB2_CKAHB_D</i> <i>IV8</i>	select CK_AHB / 8 as CK_APB2
<i>RCU_APB2_CKAHB_D</i> <i>IV16</i>	select CK_AHB / 16 as CK_APB2
<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\_ckout\_config

The description of rcu\_ckout\_config is shown as below:

**Table 3-563. Function rcu\_ckout\_config**

<b>Function name</b>	rcu_ckout_config
<b>Function prototype</b>	void rcu_ckout_config(uint32_t ckout_src)
<b>Function descriptions</b>	configure the CK_OUT clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ckout_src</b>	CK_OUT clock source selection
<i>RCU_CKOUTSRC_CK</i> <i>SYS</i>	system clock selected
<i>RCU_CKOUTSRC_IRC</i> <i>8M</i>	high speed 8M internal oscillator clock selected
<i>RCU_CKOU0SRC_HX</i> <i>TAL</i>	HXTAL selected



<i>RCU_CKOUTSRC_CK</i> <i>PLL0_DIV2</i>	CK_PLL0 / 2 selected
<i>RCU_CKOUTSRC_CK</i> <i>PLL1_DIV2</i>	CK_PLL1 / 2 selected
<i>RCU_CKOUTSRC_LXT</i> <i>AL</i>	LXTAL clock selected
<i>RCU_CKOUTSRC_IRC</i> <i>48M</i>	high speed 48M internal oscillator clock selected
<i>RCU_CKOUTSRC_IRC</i> <i>40K</i>	IRC40K clock selected
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the HXTAL as CK_OUT clock source */
```

```
rcu_ckout_config(RCU_CKOUTSRC_HXTAL);
```

### rcu\_pll0\_config

The description of rcu\_pll0\_config is shown as below:

**Table 3-564. Function rcu\_pll0\_config**

<b>Function name</b>	rcu_pll0_config
<b>Function prototype</b>	void rcu_pll0_config(uint32_t pll0_src, uint32_t pll0_mul, uint32_t pll0_div)
<b>Function descriptions</b>	configure the PLL0 clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll0_src</b>	PLL0 clock source selection
<i>RCU_PLL0SRC_IRC8</i> <i>M</i>	IRC8M clock selected as source clock of PLL0
<i>RCU_PLL0SRC_IRC48</i> <i>M</i>	IRC48M selected as source clock of PLL0
<i>RCU_PLL0SRC_HXTA</i> <i>L</i>	HXTAL selected as source clock of PLL0
<i>RCU_PLL0SRC_CKPL</i> <i>L1</i>	CKPLL1 selected as source clock of PLL0
<b>Input parameter{in}</b>	
<b>pll0_mul</b>	PLL0 clock multiplication factor
<i>RCU_PLL0_MULx</i> (x = 4,5,6..63)	PLL0 clock multiplication factor
<b>Input parameter{in}</b>	

<b>pll0_div</b>	PLL0 clock division factor
<i>RCU_PLL0_DIVx</i> ( <i>x</i> = 1..16)	PLL0 clock division factor
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL0 clock */
```

```
rcu_pll0_config(RCU_PLL0SRC_IRC48M, RCU_PLL0_MUL4, RCU_PLL0_DIV2);
```

### rcu\_pll1\_config

The description of rcu\_pll1\_config is shown as below:

**Table 3-565. Function rcu\_pll1\_config**

<b>Function name</b>	rcu_pll1_config
<b>Function prototype</b>	void rcu_pll1_config(uint32_t pll1_src, uint32_t pll1_mul)
<b>Function descriptions</b>	configure the PLL1 clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pll1_src</b>	pll1 clock source selection
<i>RCU_PLL1SRC_HXTAL</i>	HXTAL selected as source clock of PLL1
<i>RCU_PPLL1SRC_IRC48M</i>	IRC48M selected as source clock of PLL1
<b>Input parameter{in}</b>	
<b>pll11_mul</b>	PLL1 clock multiplication factor
<i>RCU_PLL1_MULx</i> ( <i>x</i> = 4,5,6..63)	PLL1 clock multiplication factor
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL1 clock */
```

```
rcu_pll1_config(RCU_PLL1SRC_HXTAL, RCU_PLL1_MUL4 );
```

### rcu\_prediv0\_config

The description of rcu\_prediv0\_config is shown as below:

**Table 3-566. Function rcu\_prediv0\_config**

<b>Function name</b>	rcu_prediv0_config
<b>Function prototype</b>	void rcu_prediv0_config(uint32_t prediv0_div)
<b>Function descriptions</b>	configure the PREDIV0 division factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>prediv0_div</b>	PREDIV0 division factor
<i>RCU_PREDIV0_DIVx</i> ( <i>x</i> = 1..16)	PREDIV0 input source clock is divided x
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PREDIV0 division factor */
rcu_prediv0_config(RCU_PREDIV0_DIV2);
```

### rcu\_prediv1\_config

The description of rcu\_prediv1\_config is shown as below:

**Table 3-567. Function rcu\_prediv1\_config**

<b>Function name</b>	rcu_prediv1_config
<b>Function prototype</b>	void rcu_prediv1_config(uint32_t prediv1_div)
<b>Function descriptions</b>	configure the PREDIV1 division factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>prediv1_div</b>	PREDIV1 division factor
<i>RCU_PREDIV1_DIVx</i> ( <i>x</i> = 1..16)	PREDIV1 input source clock is divided x
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PREDIV1 division factor */
rcu_prediv1_config(RCU_PREDIV1_DIV2);
```

## rcu\_adc\_clock\_config

The description of rcu\_adc\_clock\_config is shown as below:

**Table 3-568. 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>	ADC prescaler select CK_APB2 / 2
<i>RCU_CKADC_CKAPB2_DIV4</i>	ADC prescaler select CK_APB2 / 4
<i>RCU_CKADC_CKAPB2_DIV6</i>	ADC prescaler select CK_APB2 / 6
<i>RCU_CKADC_CKAPB2_DIV8</i>	ADC prescaler select CK_APB2 / 8
<i>RCU_CKADC_CKAPB2_DIV12</i>	ADC prescaler select CK_APB2 / 12
<i>RCU_CKADC_CKAPB2_DIV16</i>	ADC prescaler select CK_APB2 / 16
<i>RCU_CKADC_CKAHB_DIV3</i>	ADC prescaler select CK_AHB / 3
<i>RCU_CKADC_CKAHB_DIV5</i>	ADC prescaler select CK_AHB / 5
<i>RCU_CKADC_CKAHB_DIV6</i>	ADC prescaler select CK_AHB / 6
<i>RCU_CKADC_CKAHB_DIV10</i>	ADC prescaler select CK_AHB / 10
<i>RCU_CKADC_CKAHB_DIV20</i>	ADC prescaler select CK_AHB / 20
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the ADC prescaler factor */
```

```
rcu_adc_clock_config(RCU_CKADC_CKAPB2_DIV2);
```

## rcu\_usb\_clock\_config

The description of rcu\_usb\_clock\_config is shown as below:

**Table 3-569. 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 USBFS 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_DIV2</i>	USBFS prescaler select CK_PLL0 / 2
<i>RCU_CKUSB_CKPLL_DIV3</i>	USBFS prescaler select CK_PLL0 / 3
<i>RCU_CKUSB_CKPLL_DIV4</i>	USBFS prescaler select CK_PLL0 / 4
<i>RCU_CKUSB_CKPLL_DIV5</i>	USBFS prescaler select CK_PLL0 / 5
<i>RCU_CKUSB_CKPLL_DIV6</i>	USBFS prescaler select CK_PLL0 / 6
<i>RCU_CKUSB_CKPLL_DIV7</i>	USBFS prescaler select CK_PLL0 / 7
<i>RCU_CKUSB_CKPLL_DIV8</i>	USBFS prescaler select CK_PLL0 / 8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the USBFS prescaler factor */
```

```
rcu_usb_clock_config(RCU_CKUSB_CKPLL_DIV2);
```

## rcu\_rtc\_clock\_config

The description of rcu\_rtc\_clock\_config is shown as below:

**Table 3-570. 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>	-

Input parameter{in}	
<b>rtc_clock_source</b>	RTC clock source selection
<i>RCU_RTCSRC_NONE</i>	no clock selected
<i>RCU_RTCSRC_LXTAL</i>	CK_LXTAL selected as RTC source clock
<i>RCU_RTCSRC_IRC40K</i>	CK_IRC40K selected as RTC source clock
<i>RCU_RTCSRC_HXTAL_DIV_128</i>	CK_HXTAL / 128 selected as RTC source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the RTC clock source selection */
rcu_rtc_clock_config(RCU_RTCSRC_LXTAL);
```

### rcu\_i2s1\_clock\_config

The description of rcu\_i2s1\_clock\_config is shown as below:

**Table 3-571. 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, uint32_t i2s_clock_div)
<b>Function descriptions</b>	configure the I2S1 clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>i2s_clock_source</b>	I2S1 clock source selection
<i>RCU_I2S1SRC_CKSYS</i>	System clock selected as I2S1 source clock
<i>RCU_I2S1SRC_CKPLL1</i>	CK_PLL1 selected as I2S1 source clock
Input parameter{in}	
<b>i2s_clock_div</b>	i2s clock division factor
<i>RCU_I2S1_DIVx</i>	(x = 1...32)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the I2S1 clock source selection */
```

```
rcu_i2s1_clock_config(RCU_I2S1SRC_CKSYS, RCU_I2S1_DIV2);
```

### rcu\_i2s2\_clock\_config

The description of rcu\_i2s2\_clock\_config is shown as below:

**Table 3-572. 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, uint32_t i2s_clock_div)
<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>	I2S2 clock source selection
<i>RCU_I2S2SRC_CKSYS</i>	system clock selected as I2S2 source clock
<i>RCU_I2S2SRC_CKPLL1</i>	CK_PLL1 selected as I2S1 source clock
<b>Input parameter{in}</b>	
<b>i2s_clock_div</b>	i2s clock division factor
<i>RCU_I2S2_DIVx</i>	(x = 1...32)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the I2S2 clock source selection */
```

```
rcu_i2s2_clock_config(RCU_I2S2SRC_CKSYS, RCU_I2S2_DIV2 );
```

### rcu\_ck48m\_clock\_config

The description of rcu\_ck48m\_clock\_config is shown as below:

**Table 3-573. 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_CKPLL0</i>	CK_PLL0 selected as CK48M source clock

<i>RCU_CK48MSRC_IRC</i> <i>48M</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_CKPLL0);
```

### rcu\_fmc\_clock\_config

The description of rcu\_fmc\_clock\_config is shown as below:

**Table 3-574. Function rcu\_fmc\_clock\_config**

<b>Function name</b>	rcu_fmc_clock_config
<b>Function prototype</b>	void rcu_fmc_clock_config(uint32_t fmc_clock_source, uint32_t fmc_clock_div)
<b>Function descriptions</b>	configure the FMC clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>fmc_clock_source</b>	FMC clock source selection
<i>RCU_FMC_CK_AHB</i>	CK_AHB clock selected as FMC source clock
<i>RCU_FMC_CK_SYS</i>	system clock selected as FMC source clock
<i>RCU_FMC_CK_PLL0</i>	CK_PLL0 clock selected as FMC source clock
<i>RCU_FMC_CK_PLL1</i>	CK_PLL1 clock selected as FMC source clock
<b>Input parameter{in}</b>	
<b>fmc_clock_div</b>	FMC clock division factor
<i>RCU_FMC_DIVx</i>	(x = 1...16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the FMC clock source selection */
```

```
rcu_fmc_clock_config(RCU_FMC_CK_AHB, RCU_FMC_DIV2);
```

### rcu\_lxtal\_drive\_capability\_config

The description of rcu\_lxtal\_drive\_capability\_config is shown as below:



**Table 3-575. 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_HI_GHDRI</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 3-576. 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 rcu_osci_type_enum
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC8M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_IRC48M</i>	internal 48M RC oscillators(IRC48M)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLL0_CK</i>	phase locked loop 0
<i>RCU_PLL1_CK</i>	phase locked loop 1
<b>Output parameter{out}</b>	
-	-

Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* wait for oscillator stabilization flag */
if(SUCCESS == rcu_osc_stab_wait(RCU_HXTAL)){
}
```

## rcu\_osc\_on

The description of rcu\_osc\_on is shown as below:

**Table 3-577. Function rcu\_osc\_on**

Function name	rcu_osc_on
Function prototype	void rcu_osc_on(rcu_osc_type_enum osci)
Function descriptions	turn on the oscillator
Precondition	-
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to rcu_osc_type_enum
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)
RCU_IRC8M	internal 8M RC oscillators(IRC8M)
RCU_IRC48M	internal 48M RC oscillators(IRC48M)
RCU_IRC40K	internal 40K RC oscillator(IRC40K)
RCU_PLL0_CK	phase locked loop 0(PLL0)
RCU_PLL1_CK	phase locked loop 1(PLL1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn on the high speed crystal oscillator */
rcu_osc_on(RCU_HXTAL);
```

## rcu\_osc\_off

The description of rcu\_osc\_off is shown as below:

**Table 3-578. Function rcu\_osc\_off**

Function name	rcu_osc_off
Function prototype	void rcu_osc_off(rcu_osc_type_enum osci)
Function descriptions	turn off the oscillator

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to rcu_osci_type_enum
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC16M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_IRC48M</i>	internal 48M RC oscillators(IRC48M)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLL0_CK</i>	phase locked loop 0(PLL0)
<i>RCU_PLL1_CK</i>	phase locked loop 1(PLL1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn off the high speed crystal oscillator */
```

```
rcu_osci_off(RCU_HXTAL);
```

### rcu\_osci\_bypass\_mode\_enable

The description of rcu\_osci\_bypass\_mode\_enable is shown as below:

**Table 3-579. Function rcu\_osci\_bypass\_mode\_enable**

<b>Function name</b>	rcu_osci_bypass_mode_enable
<b>Function prototype</b>	void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci)
<b>Function descriptions</b>	enable the oscillator bypass mode, HXTALEN or LXTALEN must be reset before it
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to rcu_osci_type_enum
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

## rcu\_osci\_bypass\_mode\_disable

The description of rcu\_osci\_bypass\_mode\_disable is shown as below:

**Table 3-580. Function rcu\_osci\_bypass\_mode\_disable**

<b>Function name</b>	rcu_osci_bypass_mode_disable
<b>Function prototype</b>	void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci)
<b>Function descriptions</b>	disable the oscillator bypass mode, HXTALEN or LXTALEN must be reset before it
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to rcu_osci_type_enum
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the high speed crystal oscillator bypass mode */
rcu_osci_bypass_mode_disable(RCU_HXTAL);
```

## rcu\_irc8m\_adjust\_value\_set

The description of rcu\_irc8m\_adjust\_value\_set is shown as below:

**Table 3-581. Function rcu\_irc8m\_adjust\_value\_set**

<b>Function name</b>	rcu_irc8m_adjust_value_set
<b>Function prototype</b>	void rcu_irc8m_adjust_value_set(uint32_t irc8m_adjval)
<b>Function descriptions</b>	set the IRC8M adjust value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>irc8m_adjval</b>	IRC8M adjust value, must be between 0 and 0x1F
<i>0x00 - 0x1F</i>	<i>0x00 - 0x1F</i>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the IRC8M adjust value */
```

```
rcu_irc8m_adjust_value_set(0x10);
```

### rcu\_hxtal\_clock\_monitor\_enable

The description of rcu\_hxtal\_clock\_monitor\_enable is shown as below:

**Table 3-582. 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 3-583. Function rcu\_hxtal\_clock\_monitor\_disable**

<b>Function name</b>	rcu_hxtal_clock_monitor_disable
<b>Function prototype</b>	void rcu_hxtal_clock_monitor_disable(void)
<b>Function descriptions</b>	disable the HXTAL clock monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the HXTAL clock monitor */
```

```
rcu_hxtal_clock_monitor_disable();
```

## rcu\_lxtal\_clock\_monitor\_enable

The description of rcu\_lxtal\_clock\_monitor\_enable is shown as below:

**Table 3-584. Function rcu\_lxtal\_clock\_monitor\_enable**

<b>Function name</b>	rcu_lxtal_clock_monitor_enable
<b>Function prototype</b>	void rcu_lxtal_clock_monitor_enable(void)
<b>Function descriptions</b>	enable the LXTAL 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 LXTAL clock monitor */
rcu_lxtal_clock_monitor_enable();
```

## rcu\_lxtal\_clock\_monitor\_disable

The description of rcu\_lxtal\_clock\_monitor\_disable is shown as below:

**Table 3-585. Function rcu\_lxtal\_clock\_monitor\_disable**

<b>Function name</b>	rcu_lxtal_clock_monitor_disable
<b>Function prototype</b>	void rcu_lxtal_clock_monitor_disable(void)
<b>Function descriptions</b>	disable the LXTAL clock monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the LXTAL clock monitor */
rcu_lxtal_clock_monitor_disable();
```

## rcu\_clock\_freq\_monitor\_enable

The description of rcu\_clock\_freq\_monitor\_enable is shown as below:

Table 3-586. Function rcu\_clock\_freq\_monitor\_enable

<b>Function name</b>	rcu_clock_freq_monitor_enable
<b>Function prototype</b>	void rcu_clock_freq_monitor_enable(rcu_ckfm_enum ckfm)
<b>Function descriptions</b>	enable the clock frequency monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ckfm</b>	clock frequency monitor enable types, refer to rcu_ckfm_enum
<i>RCU_IRC8MCKFM</i>	IRC8M clock frequency monitor
<i>RCU_HXTALCKFM</i>	HXTAL clock frequency monitor
<i>RCU_PLL0PCKFM</i>	PLL0P clock frequency monitor
<i>RCU_PLL1CKFM</i>	PLL1 clock frequency monitor
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the IRC8M clock frequency monitor */
rcu_clock_freq_monitor_enable(RCU_IRC8MCKFM);
```

### rcu\_clock\_freq\_monitor\_disable

The description of rcu\_clock\_freq\_monitor\_disable is shown as below:

Table 3-587. Function rcu\_clock\_freq\_monitor\_disable

<b>Function name</b>	rcu_clock_freq_monitor_disable
<b>Function prototype</b>	void rcu_clock_freq_monitor_disable(rcu_ckfm_enum ckfm)
<b>Function descriptions</b>	disable the clock frequency monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ckfm</b>	clock frequency monitor enable types, refer to rcu_ckfm_enum
<i>RCU_IRC8MCKFM</i>	IRC8M clock frequency monitor
<i>RCU_HXTALCKFM</i>	HXTAL clock frequency monitor
<i>RCU_PLL0PCKFM</i>	PLL0P clock frequency monitor
<i>RCU_PLL1CKFM</i>	PLL1 clock frequency monitor
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the IRC8M clock frequency monitor */
```

```
rcu_clock_freq_monitor_disable(RCU_IRC8MCKFM);
```

### rcu\_irc8m\_freq\_monitor\_config

The description of rcu\_irc8m\_freq\_monitor\_config is shown as below:

**Table 3-588. Function rcu\_irc8m\_freq\_monitor\_config**

Function name	rcu_irc8m_freq_monitor_config
Function prototype	void rcu_irc8m_freq_monitor_config(uint32_t range)
Function descriptions	IRC8M clock frequency monitor range configuration
Precondition	-
The called functions	-
Input parameter{in}	
range	IRC8M clock frequency monitor configuration
RCU_IRC8MCKFMC_5_PERCENT	the IRC8M clock frequency monitoring range is $\pm 5\%$
RCU_IRC8MCKFMC_10_PERCENT	the IRC8M clock frequency monitoring range is $\pm 10\%$
RCU_IRC8MCKFMC_15_PERCENT	the IRC8M clock frequency monitoring range is $\pm 15\%$
RCU_IRC8MCKFMC_20_PERCENT	the IRC8M clock frequency monitoring range is $\pm 20\%$
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* IRC8M clock frequency monitor range configuration */
rcu_irc8m_freq_monitor_config(RCU_IRC8MCKFMC_5_PERCENT);
```

### rcu\_hxtal\_monitor\_threshold\_config

The description of rcu\_hxtal\_monitor\_threshold\_config is shown as below:

**Table 3-589. Function rcu\_hxtal\_monitor\_threshold\_config**

Function name	rcu_hxtal_monitor_threshold_config
Function prototype	void rcu_hxtal_monitor_threshold_config(uint32_t lthreshold, uint32_t hthreshold)
Function descriptions	HXTAL clock frequency monitor threshold configuration
Precondition	-
The called functions	-
Input parameter{in}	
lthreshold	low frequency threshold configuration
0x00 - 0xfff	low frequency threshold



Input parameter{in}	
<b>hthreshold</b>	high frequency threshold configuration
<i>0x00 - 0xff</i>	high frequency threshold
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* HXTAL clock frequency monitor threshold configuration */
```

```
rcu_hxtal_monitor_threshold_config(0x07f, 07ff);
```

### rcu\_pll0p\_monitor\_threshold\_config

The description of rcu\_pll0p\_monitor\_threshold\_config is shown as below:

**Table 3-590. Function rcu\_pll0p\_monitor\_threshold\_config**

<b>Function name</b>	rcu_pll0p_monitor_threshold_config
<b>Function prototype</b>	void rcu_pll0p_monitor_threshold_config(uint32_t lthreshold, uint32_t hthreshold)
<b>Function descriptions</b>	PLL0P clock frequency monitor threshold configuration
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>lthreshold</b>	low frequency threshold configuration
<i>0x00 - 0x3ff</i>	low frequency threshold
Input parameter{in}	
<b>hthreshold</b>	high frequency threshold configuration
<i>0x00 - 0x3ff</i>	high frequency threshold
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PLL0P clock frequency monitor threshold configuration */
```

```
rcu_pll0p_monitor_threshold_config(0x07f, 0x300);
```

### rcu\_pll1\_monitor\_threshold\_config

The description of rcu\_pll1\_monitor\_threshold\_config is shown as below:

**Table 3-591. Function rcu\_pll1\_monitor\_threshold\_config**

<b>Function name</b>	rcu_pll1_monitor_threshold_config
----------------------	-----------------------------------

<b>Function prototype</b>	void rcu_pll1_monitor_threshold_config(uint32_t lthreshold, uint32_t hthreshold)
<b>Function descriptions</b>	PLL1 clock frequency monitor threshold configuration
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lthreshold</b>	low frequency threshold configuration
0x00 - 0x3ff	low frequency threshold
<b>Input parameter{in}</b>	
<b>hthreshold</b>	high frequency threshold configuration
0x00 - 0x3ff	high frequency threshold
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PLL1 clock frequency monitor threshold configuration */
```

```
rcu_pll1_monitor_threshold_config(0x07f, 0x300);
```

### rcu\_deepsleep\_voltage\_set

The description of rcu\_deepsleep\_voltage\_set is shown as below:

**Table 3-592. 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
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dsvol</b>	deep sleep mode voltage
RCU_DEEPSLEEP_V_0	the core voltage is default value
RCU_DEEPSLEEP_V_1	the core voltage is (default value-0.05)V
RCU_DEEPSLEEP_V_2	the core voltage is (default value-0.1)V
RCU_DEEPSLEEP_V_3	the core voltage is (default value-0.15)V
RCU_DEEPSLEEP_V_4	the core voltage is (default value-0.2)V
RCU_DEEPSLEEP_V_5	the core voltage is (default value-0.25)V

RCU_DEEPSLEEP_V_6	the core voltage is (default value-0.3)V(customers are not recommended to use it)
RCU_DEEPSLEEP_V_7	the core voltage is (default value-0.35)V(customers are not recommended to use it)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the deep sleep mode voltage */
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_0);
```

### rcu\_deepsleep\_switch\_delay\_set

The description of rcu\_deepsleep\_switch\_delay\_set is shown as below:

**Table 3-593. Function rcu\_deepsleep\_switch\_delay\_set**

Function name	rcu_deepsleep_switch_delay_set
Function prototype	void rcu_deepsleep_switch_delay_set(uint32_t irc8m_cnt)
Function descriptions	delay before switch to IRC8M clock and enter deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
hsi_cnt	1 ~ 31
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* delay before switch to IRC8M clock and enter deep-sleep mode */
rcu_deepsleep_switch_delay_set(5);
```

### rcu\_reg\_lock

The description of rcu\_reg\_lock is shown as below:

**Table 3-594. Function rcu\_reg\_lock**

Function name	rcu_reg_lock
Function prototype	void rcu_reg_lock(void)
Function descriptions	lock rcu register
Precondition	-
The called functions	-

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

Example:

```
/* lock rcu register */
```

```
rcu_reg_lock();
```

### rcu\_reg\_unlock

The description of rcu\_reg\_unlock is shown as below:

**Table 3-595. Function rcu\_reg\_unlock**

Function name	rcu_reg_unlock
Function prototype	void rcu_reg_unlock(void)
Function descriptions	unlock rcu register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock rcu register */
```

```
rcu_reg_unlock();
```

### rcu\_pll\_bandwidth\_config

The description of rcu\_pll\_bandwidth\_config is shown as below:

**Table 3-596. Function rcu\_pll\_bandwidth\_config**

Function name	rcu_pll_bandwidth_config
Function prototype	void rcu_pll_bandwidth_config(uint32_t pll0_bw, uint32_t pll1_bw)
Function descriptions	config PLL0/PLL1 bandwidth
Precondition	-
The called functions	-
Input parameter{in}	
pll0_bw	pll0 bandwidth configuration, must be between 0 and 0xF

0 ~ 15	pll0 bandwidth
<b>Input parameter{in}</b>	
<b>pll1_bw</b>	pll1 bandwidth configuration, must be between 0 and 0xF
0 ~ 15	pll1 bandwidth
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config PLL0/PLL1 bandwidth */
rcu_pll_bandwidth_config(0, 0);
```

### rcu\_clock\_freq\_get

The description of rcu\_clock\_freq\_get is shown as below:

**Table 3-597. 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 and peripheral clock frequency
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>clock</b>	the clock frequency which to get
CK_SYS	system clock frequency
CK_AHB	AHB clock frequency
CK_APB1	APB1 clock frequency
CK_APB2	APB2 clock frequency
CK_PLL0	PLL0 clock frequency
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	clock frequency of system, AHB, APB1, APB2, PLL0

Example:

```
uint32_t temp_freq;

/* get the system clock frequency */
temp_freq = rcu_clock_freq_get(CK_SYS);
```

### rcu\_flag\_get

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

Table 3-598. 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, peripheral reset and clock frequency failure 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 rcu_flag_enum
<i>RCU_FLAG_IRC8MST</i> <i>B</i>	IRC8M stabilization flag
<i>RCU_FLAG_HXTALST</i> <i>B</i>	HXTAL stabilization flag
<i>RCU_FLAG_PLL0STB</i>	PLL0 stabilization flag
<i>RCU_FLAG_PLL1STB</i>	PLL1 stabilization flag(CL series only)
<i>RCU_FLAG_LXTALST</i> <i>B</i>	LXTAL stabilization flag
<i>RCU_FLAG_IRC40KST</i> <i>B</i>	IRC40K stabilization flag
<i>RCU_FLAG_IRC48MS</i> <i>TB</i>	IRC48M stabilization flag
<i>RCU_FLAG_EPRST</i>	external PIN reset flag
<i>RCU_FLAG_PORRST</i>	power reset flag
<i>RCU_FLAG_SWRST</i>	software reset flag
<i>RCU_FLAG_FWDGTR</i> <i>ST</i>	free watchdog timer reset flag
<i>RCU_FLAG_WWDGTR</i> <i>ST</i>	window watchdog timer reset flag
<i>RCU_FLAG_LPRST</i>	low-power reset flag
<i>RCU_FLAG_IRC8MCK</i> <i>FF</i>	IRC8M clock frequency failure flag
<i>RCU_FLAG_HXTALCK</i> <i>FF</i>	HXTAL clock frequency failure flag
<i>RCU_FLAG_PLL0PCK</i> <i>FF</i>	PLL0P clock frequency failure flag
<i>RCU_FLAG_PLL1CKF</i> <i>F</i>	PLL1 clock frequency failure flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	status of flag (RESET or SET)

Example:

```

/* get the clock stabilization, peripheral reset and clock frequency failure flags */

/* get the clock stabilization interrupt flag */

if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){

};

```

### rcu\_flag\_clear

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

**Table 3-599. Function rcu\_flag\_clear**

Function name	rcu_flag_clear
Function prototype	void rcu_flag_clear(rcu_flag_clear_enum flag)
Function descriptions	clear clock frequency failure flags
Precondition	-
The called functions	-
Input parameter{in}	
flag	clock frequency failure flag, refer to rcu_flag_clear_enum
<i>RCU_FLAG_IRC8MCK FF_CLR</i>	IRC8M clock frequency failure interrupt flag clear
<i>RCU_FLAG_HXTALCK FF_CLR</i>	HXTAL clock frequency failure interrupt flag clear
<i>RCU_FLAG_PLL0PCK FF_CLR</i>	PLL0P clock frequency failure interrupt flag clear
<i>RCU_FLAG_PLL1CKF F_CLR</i>	PLL1 clock frequency failure interrupt flag clear
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* clear the interrupt HXTAL stabilization interrupt flag */

rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);

```

### rcu\_all\_reset\_flag\_clear

The description of rcu\_all\_reset\_flag\_clear is shown as below:

**Table 3-600. Function rcu\_all\_reset\_flag\_clear**

Function name	rcu_all_reset_flag_clear
Function prototype	void rcu_all_reset_flag_clear(void)
Function descriptions	clear all the reset flag
Precondition	-

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

Example:

```
/* clear all the reset flag */
```

```
rcu_all_reset_flag_clear();
```

### rcu\_interrupt\_enable

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

**Table 3-601. Function rcu\_interrupt\_enable**

Function name	rcu_interrupt_enable
Function prototype	void rcu_interrupt_enable(rcu_int_enum interrupt)
Function descriptions	enable the clock interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	interrupt clock stabilization interrupt, refer to rcu_int_enum
RCU_INT_IRC40KSTB	IRC40K stabilization interrupt enable
RCU_INT_LXTALSTB	LXTAL stabilization interrupt enable
RCU_INT_IRC8MSTB	IRC8M stabilization interrupt enable
RCU_INT_HXTALSTB	HXTAL stabilization interrupt enable
RCU_INT_PLL0STB	PLL0 stabilization interrupt enable
RCU_INT_PLL1STB	PLL1 stabilization interrupt enable
RCU_INT_IRC48MSTB	IRC48M stabilization interrupt enable
RCU_INT_IRC8MCKFF	IRC8M clock frequency failure interrupt enable
RCU_INT_HXTALCKFF	HXTAL clock frequency failure interrupt enable
RCU_INT_PLL0PCKFF	PLL0P clock frequency failure interrupt enable
RCU_INT_PLL1CKFF	PLL1 clock frequency failure interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the HXTAL stabilization interrupt */
```

```
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```



## rcu\_interrupt\_disable

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

**Table 3-602. 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 clock interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
interrupt	interrupt clock stabilization interrupt, refer to rcu_int_enum
RCU_INT_IRC40KSTB	IRC40K stabilization interrupt disable
RCU_INT_LXTALSTB	LXTAL stabilization interrupt disable
RCU_INT_IRC8MSTB	IRC8M stabilization interrupt disable
RCU_INT_HXTALSTB	HXTAL stabilization interrupt disable
RCU_INT_PLL0STB	PLL0 stabilization interrupt disable
RCU_INT_PLL1STB	PLL1 stabilization interrupt disable
RCU_INT_IRC48MSTB	IRC48M stabilization interrupt disable
RCU_INT_IRC8MCKFF	IRC8M clock frequency failure interrupt disable
RCU_INT_HXTALCKFF	HXTAL clock frequency failure interrupt disable
RCU_INT_PLL0PCKFF	PLL0P clock frequency failure interrupt disable
RCU_INT_PLL1CKFF	PLL1 clock frequency failure interrupt disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

## rcu\_interrupt\_flag\_get

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

**Table 3-603. 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 interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
int_flag	interrupt and clock stuck flags, refer to rcu_int_flag_enum

<i>RCU_INT_FLAG_IRC4 OKSTB</i>	IRC40K stabilization interrupt flag
<i>RCU_INT_FLAG_LXTA LSTB</i>	LXTAL stabilization interrupt flag
<i>RCU_INT_FLAG_IRC8 MSTB</i>	IRC8M stabilization interrupt flag
<i>RCU_INT_FLAG_HXTA LSTB</i>	HXTAL stabilization interrupt flag
<i>RCU_INT_FLAG_PLL0 STB</i>	PLL0 stabilization interrupt flag
<i>RCU_INT_FLAG_PLL1 STB</i>	PLL1 stabilization interrupt flag
<i>RCU_INT_FLAG_HCK M</i>	HXTAL clock stuck interrupt flag
<i>RCU_INT_FLAG_LCK M</i>	LXTAL clock stuck interrupt flag
<i>RCU_INT_FLAG_IRC4 8MSTB</i>	IRC48M stabilization interrupt flag
<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 3-604. 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 clock 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 rcu_int_flag_clear_enum
<i>RCU_INT_FLAG_IRC4 OKSTB_CLR</i>	IRC40K stabilization interrupt flag clear

<i>RCU_INT_FLAG_LXTA LSTB_CLR</i>	LXTAL stabilization interrupt flag clear
<i>RCU_INT_FLAG_IRC8 MSTB_CLR</i>	IRC8M stabilization interrupt flag clear
<i>RCU_INT_FLAG_HXTA LSTB_CLR</i>	HXTAL stabilization interrupt flag clear
<i>RCU_INT_FLAG_PLL0 STB_CLR</i>	PLL0 stabilization interrupt flag clear
<i>RCU_INT_FLAG_PLL1 STB_CLR</i>	PLL1 stabilization interrupt flag clear
<i>RCU_INT_FLAG_HCK M_CLR</i>	HXTAL clock stuck interrupt flag clear
<i>RCU_INT_FLAG_LCK M_CLR</i>	LXTAL clock stuck interrupt flag clear
<i>RCU_INT_FLAG_IRC4 8MSTB_CLR</i>	IRC48M stabilization interrupt flag clear
<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);
```

## 3.22. RTC

The Real-time Clock (RTC) is usually used as a clock-calendar. The ones in the Backup Domain consist of a 32-bit up-counter, an alarm, a prescaler, a divider and the RTC clock configuration register. The RTC registers are listed in chapter [3.22.1](#), the RTC firmware functions are introduced in chapter [3.22.2](#).

### 3.22.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

**Table 3-605. RTC Registers**

Registers	Descriptions
RTC_INTEN	Interrupt enable register
RTC_CTL	Control register
RTC_PSCH	Prescaler high register

Registers	Descriptions
RTC_PSCL	Prescaler low register
RTC_DIVH	Divider high register
RTC_DIVL	Divider low register
RTC_CNTH	counter high register
RTC_CNTL	counter low register
RTC_ALRMH	Alarm high register
RTC_ALRML	Alarm low register

### 3.22.2. Descriptions of Peripheral functions

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

**Table 3-606. 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
rtc_flag_clear	clear RTC flag status

#### rtc\_configuration\_mode\_enter

The description of rtc\_configuration\_mode\_enter is shown as below:

**Table 3-607. Function rtc\_configuration\_mode\_enter**

Function name	rtc_configuration_mode_enter
Function prototype	void rtc_configuration_mode_enter(void);
Function descriptions	enter RTC configuration mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enter RTC configuration mode */
```

```
rtc_configuration_mode_enter();
```

### rtc\_configuration\_mode\_exit

The description of rtc\_configuration\_mode\_exit is shown as below:

**Table 3-608. 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 3-609. 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 3-610. 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 3-611. 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}	

-	-
<b>Return value</b>	
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 3-612. Function rtc\_counter\_set**

<b>Function name</b>	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>	
cnt	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 3-613. 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

Input parameter{in}	
<b>psc</b>	RTC prescaler value (0-0x000F FFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait();
```

```
/* set RTC prescaler value to 0x7FFFF */
```

```
rtc_prescaler_set(0x7FFFF);
```

### rtc\_alarm\_config

The description of rtc\_alarm\_config is shown as below:

**Table 3-614. 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
Input parameter{in}	
<b>alarm</b>	RTC alarm value(0-0xFFFF FFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait();
```

```
/* set alarm value to 0xFFFF */
```

```
rtc_alarm_config (0xFFFF);
```

### rtc\_divider\_get

The description of rtc\_divider\_get is shown as below:



Table 3-615. Function rtc\_divider\_get

Function name	rtc_divider_get
Function prototype	uint32_t rtc_divider_get(void);
Function descriptions	get RTC divider value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the value of RTC divider

Example:

```
/* get the current RTC divider value */
uint32_t rtc_divider_value;
rtc_divider_value = rtc_divider_get();
```

### rtc\_interrupt\_enable

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

Table 3-616. 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 3-617. Function rtc\_interrupt\_disable**

<b>Function name</b>	rtc_interrupt_disable
<b>Function prototype</b>	void rtc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable RTC interrupt
<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>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which RTC interrupt to disable
RTC_INT_SECOND	second interrupt
RTC_INT_ALARM	alarm interrupt
RTC_INT_OVERFLOW	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\_flag\_get

The description of rtc\_flag\_getrtc\_interrupt\_enable is shown as below:

**Table 3-618. 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>	
<b>flag</b>	specify which RTC flag status to get
RTC_FLAG_SECOND	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 3-619. 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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the RTC alarm flag */
```

```
rtc_flag_clear(RTC_FLAG_ALARM);
```

## 3.23. SYSCFG

The SYSCFG registers are listed in chapter [3.23.1](#), and the SYSCFG firmware functions are introduced in chapter [3.23.2](#).

### 3.23.1. Descriptions of Peripheral registers

SYSCFG registers are listed in the table shown as below:

**Table 3-620. SYSCFG Registers**

Registers	Descriptions
SYSCFG_CFG0	SYSCFG configuration register 0
SYSCFG_CFG1	SYSCFG configuration register 1
SYSCFG_LKCTL	SYSCFG lockup control register
SYSCFG_BUSTO	SYSCFG bus timeout register
SYSCFG_TIMERCISEL	SYSCFG timer input selection register
SYSCFG_FPUINTEN	SYSCFG FPU interrupt enable register
SYSCFG_SRAMWP	SYSCFG SRAM write protection register
SYSCFG_SRAM ECCSTAT	SYSCFG SRAM ECC status register
SYSCFG_SRAM ECCCS	SYSCFG SRAM ECC control and status register
SYSCFG_BUSTO STAT	SYSCFG bus timeout status register
SYSCFG_TIMER0TRGCFG0	Timer0 TRG configuration register 0
SYSCFG_TIMER0TRGCFG1	Timer0 TRG configuration register 1
SYSCFG_TIMER0TRGCFG2	Timer0 TRG configuration register 2
SYSCFG_TIMER1TRGCFG0	Timer1 TRG configuration register 0
SYSCFG_TIMER1TRGCFG1	Timer1 TRG configuration register 1
SYSCFG_TIMER1TRGCFG2	Timer1 TRG configuration register 2
SYSCFG_TIMER2TRGCFG0	Timer2 TRG configuration register 0
SYSCFG_TIMER2TRGCFG1	Timer2 TRG configuration register 1
SYSCFG_TIMER2TRGCFG2	Timer2 TRG configuration register 2
SYSCFG_TIMER3TRGCFG0	Timer3 TRG configuration register 0
SYSCFG_TIMER3TRGCFG1	Timer3 TRG configuration register 1
SYSCFG_TIMER3TRGCFG2	Timer3 TRG configuration register 2
SYSCFG_TIMER4TRGCFG0	Timer4 TRG configuration register 0
SYSCFG_TIMER4TRGCFG1	Timer4 TRG configuration register 1
SYSCFG_TIMER4TRGCFG2	Timer4 TRG configuration register 2
SYSCFG_TIMER7TRGCFG0	Timer7 TRG configuration register 0
SYSCFG_TIMER7TRGCFG1	Timer7 TRG configuration register 1
SYSCFG_TIMER7TRGCFG2	Timer7 TRG configuration register 2
SYSCFG_TIMER15TRGCFG0	Timer15 TRG configuration register0
SYSCFG_TIMER15TRGCFG1	Timer15 TRG configuration register 1
SYSCFG_TIMER15TRGCFG2	Timer15 TRG configuration register 2
SYSCFG_TIMER16TRGCFG0	Timer16 TRG configuration register 0

Registers	Descriptions
SYSCFG_TIMER16TRGCFG1	Timer16 TRG configuration register 1
SYSCFG_TIMER16TRGCFG2	Timer16 TRG configuration register 2

### 3.23.2. Descriptions of Peripheral functions

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

**Table 3-621. SYSCFG firmware function**

Function name	Function description
syscfg_deinit	reset the SYSCFG registers
syscfg_bootmode_get	get current boot mode
syscfg_i2c_fast_mode_plus_enable	enable I2Cx Fast Mode Plus
syscfg_i2c_fast_mode_plus_disable	disable I2Cx Fast Mode Plus
syscfg_lockup_enable	enable module lockup function
syscfg_bus_timeout_enable	enable bus timeout
syscfg_bus_timeout_disable	disable bus timeout
syscfg_timer_input_source_select	Select TIMER channel input source
syscfg_sram_page_wp_enable	enable SRAM page write protection
syscfg_sram_ecc_error_bit_get	get SRAM ECC single-bit error bit position
syscfg_sram_ecc_error_addr_get	get SRAM ECC error address
syscfg_fpu_interrupt_enable	enable FPU interrupt
syscfg_fpu_interrupt_disable	disable FPU interrupt
syscfg_sram_ecc_interrupt_enable	enable SRAM ECC error interrupt
syscfg_sram_ecc_interrupt_disable	disable SRAM ECC error interrupt
syscfg_bus_timeout_flag_get	get bus timeout flag
syscfg_bus_timeout_flag_clear	clear bus timeout flag
syscfg_sram_ecc_flag_get	get SRAM ECC error flag
syscfg_sram_ecc_flag_clear	clear SRAM ECC error flag

### Enum timer\_channel\_input\_enum

**Table 3-622. Enum timer\_channel\_input\_enum**

enum name	Function description
TIMER15_CIO_INPU T_TIMER15_CH0	Select TIMER15_CH0 as TIMER15 CIO
TIMER15_CIO_INPU T_IRC40K	Select IRC40K as TIMER15 CIO
TIMER15_CIO_INPU T_LXTAL	Select LXTAL as TIMER15 CIO
TIMER16_CIO_INPU T_TIMER16_CH0	Select TIMER16_CH0 as TIMER16 CIO
TIMER16_CIO_INPU T_CKOUT	Select CKOUT as TIMER16 CIO

## syscfg\_deinit

The description of syscfg\_deinit is shown as below:

**Table 3-623. Function syscfg\_deinit**

<b>Function name</b>	syscfg_deinit
<b>Function prototype</b>	void syscfg_deinit(void)
<b>Function descriptions</b>	reset the SYSCFG registers
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset SYSCFG registers */
syscfg_deinit();
```

## syscfg\_bootmode\_select

The description of syscfg\_bootmode\_select is shown as below:

**Table 3-624. Function syscfg\_bootmode\_select**

<b>Function name</b>	syscfg_bootmode_select
<b>Function prototype</b>	void syscfg_bootmode_select(uint32_t bootmode)
<b>Function descriptions</b>	select bootmode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>bootmode</b>	<b>bootmode</b>
SYSCFG_BOOTMODE_FLASH	boot from main flash
SYSCFG_BOOTMODE_BOOTLOADER	boot from bootloader
SYSCFG_BOOTMODE_SRAM	boot from embedded SRAM
SYSCFG_BOOTMODE_OTP1	boot from embedded OTP1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* select boot from main flash */
syscfg_bootmode_select(SYSCFG_BOOTMODE_FLASH);
```

### syscfg\_i2c\_fast\_mode\_plus\_enable

The description of syscfg\_i2c\_fast\_mode\_plus\_enable is shown as below:

**Table 3-625. Function syscfg\_i2c\_fast\_mode\_plus\_enable**

<b>Function name</b>	syscfg_i2c_fast_mode_plus_enable
<b>Function prototype</b>	void syscfg_i2c_fast_mode_plus_enable(uint32_t i2c_fmp)
<b>Function descriptions</b>	enable I2Cx Fast Mode Plus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_fmp</b>	I2C fast mode plus
<i>SYSCFG_I2C0_FMP</i>	I2C0 fast mode plus
<i>SYSCFG_I2C1_FMP</i>	I2C1 fast mode plus
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 fast mode plus function */
syscfg_i2c_fast_mode_plus_enable(SYSCFG_I2C0_FMP);
```

### syscfg\_i2c\_fast\_mode\_plus\_disable

The description of syscfg\_i2c\_fast\_mode\_plus\_disable is shown as below:

**Table 3-626. Function syscfg\_i2c\_fast\_mode\_plus\_disable**

<b>Function name</b>	syscfg_i2c_fast_mode_plus_disable
<b>Function prototype</b>	void syscfg_i2c_fast_mode_plus_disable(uint32_t i2c_fmp)
<b>Function descriptions</b>	disable I2Cx Fast Mode Plus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_fmp</b>	I2C fast mode plus
<i>SYSCFG_I2C0_FMP</i>	I2C0 fast mode plus
<i>SYSCFG_I2C1_FMP</i>	I2C1 fast mode plus

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 fast mode plus function */
syscfg_i2c_fast_mode_plus_disable(SYSCFG_I2C0_FMP);
```

### syscfg\_lockup\_enable

The description of syscfg\_lockup\_enable is shown as below:

**Table 3-627. Function syscfg\_lockup\_enable**

Function name	syscfg_lockup_enable
Function prototype	void syscfg_lockup_enable(uint32_t lockup)
Function descriptions	enable module lockup function (function can be disabled by system reset)
Precondition	-
The called functions	-
Input parameter{in}	
lockup	lockup configuration
SYSCFG_CPU_LOCKUP	CPU lockup signal
SYSCFG_SRAM_LOCKUP	SRAM ECC double error signal
SYSCFG_LVD_LOCKUP	LVD signal connected
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable module lockup function */
syscfg_lockup_enable(SYSCFG_CPU_LOCKUP);
```

### syscfg\_bus\_timeout\_enable

The description of syscfg\_bus\_timeout\_enable is shown as below:

**Table 3-628. Function syscfg\_bus\_timeout\_enable**

Function name	syscfg_bus_timeout_enable
Function prototype	void syscfg_bus_timeout_enable(uint32_t busto)
Function descriptions	enable bus timeout



<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>busto</b>	bus timeout
<i>SYSCFG_BUSTO_CPU CBUSTO</i>	CPU Cbus timeout enable bit
<i>SYSCFG_BUSTO_CPU SBUSTO</i>	CPU Sbus timeout enable bit
<i>SYSCFG_BUSTO_DM A0BUSTO</i>	DMA0 bus timeout enable bit
<i>SYSCFG_BUSTO_DM A1BUSTO</i>	DMA1 bus timeout enable bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable CPU Cbus timeout */
```

```
syscfg_bus_timeout_enable(SYSCFG_BUSTO_CPUCBUSTO);
```

### syscfg\_bus\_timeout\_disable

The description of syscfg\_bus\_timeout\_disable is shown as below:

**Table 3-629. Function syscfg\_bus\_timeout\_disable**

<b>Function name</b>	syscfg_bus_timeout_disable
<b>Function prototype</b>	void syscfg_bus_timeout_disable(uint32_t busto)
<b>Function descriptions</b>	disable bus timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>busto</b>	bus timeout
<i>SYSCFG_CPUCBUS_T IMEROUT</i>	CPU Cbus timeout enable bit
<i>SYSCFG_CPUSBUS_T IMEROUT</i>	CPU Sbus timeout enable bit
<i>SYSCFG_DMA0BUS_T IMEROUT</i>	DMA0 bus timeout enable bit
<i>SYSCFG_DMA1BUS_T IMEROUT</i>	DMA1 bus timeout enable bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* disable CPU Cbus timeout */
```

```
syscfg_bus_timeout_disable(SYSCFG_BUSTO_CPUCBUSTO);
```

### syscfg\_timer\_input\_source\_select

The description of syscfg\_timer\_input\_source\_select is shown as below:

**Table 3-630. Function syscfg\_timer\_input\_source\_select**

Function name	syscfg_timer_input_source_select
Function prototype	void syscfg_timer_input_source_select(timer_channel_input_enum timer_input)
Function descriptions	Select TIMER channel input source
Precondition	-
The called functions	-
Input parameter{in}	
timer_input	TIMER channel input selection, refer to timer_channel_input_enum
TIMER15_CIO_INPUT_TIMER15_CH0	TIMER15_CH0 input
TIMER15_CIO_INPUT_IRC40K	IRC40K
TIMER15_CIO_INPUT_LXTAL	LXTAL
TIMER16_CIO_INPUT_TIMER16_CH0	TIMER16_CH0 input
TIMER16_CIO_INPUT_CKOUT	CKOUT
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select timer channel input source */
```

```
syscfg_timer_input_source_select(TIMER15_CIO_INPUT_IRC40K);
```

### syscfg\_sram\_page\_wp\_enable

The description of syscfg\_sram\_page\_wp\_enable is shown as below:

**Table 3-631. Function syscfg\_sram\_page\_wp\_enable**

Function name	syscfg_sram_page_wp_enable
---------------	----------------------------

<b>Function prototype</b>	void syscfg_sram_page_wp_enable(uint32_t page)
<b>Function descriptions</b>	enable SRAM page write protection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>page</b>	SRAM page
SYSCFG_SRAM_WP_PAGE0	SRAM page 0 write protection
SYSCFG_SRAM_WP_PAGE1	SRAM page 1 write protection
SYSCFG_SRAM_WP_PAGE2	SRAM page 2 write protection
SYSCFG_SRAM_WP_PAGE3	SRAM page 3 write protection
SYSCFG_SRAM_WP_PAGE4	SRAM page 4 write protection
SYSCFG_SRAM_WP_PAGE5	SRAM page 5 write protection
SYSCFG_SRAM_WP_PAGE6	SRAM page 6 write protection
SYSCFG_SRAM_WP_PAGE7	SRAM page 7 write protection
SYSCFG_SRAM_WP_PAGE8	SRAM page 8 write protection
SYSCFG_SRAM_WP_PAGE9	SRAM page 9 write protection
SYSCFG_SRAM_WP_PAGE10	SRAM page 10 write protection
SYSCFG_SRAM_WP_PAGE11	SRAM page 11 write protection
SYSCFG_SRAM_WP_PAGE12	SRAM page 12 write protection
SYSCFG_SRAM_WP_PAGE13	SRAM page 13 write protection
SYSCFG_SRAM_WP_PAGE14	SRAM page 14 write protection
SYSCFG_SRAM_WP_PAGE15	SRAM page 15 write protection
SYSCFG_SRAM_WP_PAGE16	SRAM page 16 write protection
SYSCFG_SRAM_WP_PAGE17	SRAM page 17 write protection
SYSCFG_SRAM_WP_PAGE18	SRAM page 18 write protection

<i>PAGE18</i>	
<i>SYSCFG_SRAM_WP_</i> <i>PAGE19</i>	SRAM page 19 write protection
<i>SYSCFG_SRAM_WP_</i> <i>PAGE20</i>	SRAM page 20 write protection
<i>SYSCFG_SRAM_WP_</i> <i>PAGE21</i>	SRAM page 21 write protection
<i>SYSCFG_SRAM_WP_</i> <i>PAGE22</i>	SRAM page 22 write protection
<i>SYSCFG_SRAM_WP_</i> <i>PAGE23</i>	SRAM page 23 write protection
<i>SYSCFG_SRAM_WP_</i> <i>PAGE24</i>	SRAM page 24 write protection
<i>SYSCFG_SRAM_WP_</i> <i>PAGE25</i>	SRAM page 25 write protection
<i>SYSCFG_SRAM_WP_</i> <i>PAGE26</i>	SRAM page 26 write protection
<i>SYSCFG_SRAM_WP_</i> <i>PAGE27</i>	SRAM page 27 write protection
<i>SYSCFG_SRAM_WP_</i> <i>PAGE28</i>	SRAM page 28 write protection
<i>SYSCFG_SRAM_WP_</i> <i>PAGE29</i>	SRAM page 29 write protection
<i>SYSCFG_SRAM_WP_</i> <i>PAGE30</i>	SRAM page 30 write protection
<i>SYSCFG_SRAM_WP_</i> <i>PAGE31</i>	SRAM page 31 write protection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SRAM page 0 write protection */
syscfg_sram_page_wp_enable(SYSCFG_SRAMWP_SRAM_P0WP);
syscfg_sram_page_wp_enable();
```

### syscfg\_sram\_ecc\_error\_bit\_get

The description of syscfg\_sram\_ecc\_error\_bit\_get is shown as below:

**Table 3-632. Function syscfg\_sram\_ecc\_error\_bit\_get**

<b>Function name</b>	syscfg_sram_ecc_error_bit_get
<b>Function prototype</b>	uint32_t syscfg_sram_ecc_error_bit_get(void)

<b>Function descriptions</b>	get SRAM ECC single-bit error bit position
<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>	error bit position (0-38, 0 means no error)

Example:

```
/* get SRAM ECC single-bit error bit position (API_ID(0x000AU)) */
uint32_t ecc_bit;
ecc_bit = syscfg_sram_ecc_error_bit_get();
```

### syscfg\_sram\_ecc\_error\_addr\_get

The description of syscfg\_sram\_ecc\_error\_addr\_get is shown as below:

**Table 3-633. Function syscfg\_sram\_ecc\_error\_addr\_get**

<b>Function name</b>	syscfg_sram_ecc_error_addr_get
<b>Function prototype</b>	uint32_t syscfg_sram_ecc_error_addr_get(void)
<b>Function descriptions</b>	get SRAM ECC error address
<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>	faulting system address where the last ECC event on SRAM occurred

Example:

```
/* get SRAM ECC error address */
uint32_t addr;
addr = syscfg_sram_ecc_error_addr_get();
```

### syscfg\_fpu\_interrupt\_enable

The description of syscfg\_fpu\_interrupt\_enable is shown as below:

**Table 3-634. Function syscfg\_fpu\_interrupt\_enable**

<b>Function name</b>	syscfg_fpu_interrupt_enable
<b>Function prototype</b>	void syscfg_fpu_interrupt_enable(uint32_t interrupt)

<b>Function descriptions</b>	enable FPU interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	FPU interrupt
<i>SYSCFG_FPUINT_INV ALID_OPERATION</i>	invalid operation interrupt enable bit
<i>SYSCFG_FPUINT_DIV 0</i>	divide by 0 interrupt enable bit
<i>SYSCFG_FPUINT_UN DERFLOW</i>	underflow interrupt enable bit
<i>SYSCFG_FPUINT_OV ERFLOW</i>	overflow interrupt enable bit
<i>SYSCFG_FPUINT_INP UT_ABNORMAL</i>	input abnormal interrupt enable bit
<i>SYSCFG_FPUINT_INE XACT</i>	inexact interrupt enable bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable FPU inexact interrupt */
```

```
syscfg_fpu_interrupt_enable(SYSCFG_FPUINT_INEXACT);
```

### syscfg\_fpu\_interrupt\_disable

The description of syscfg\_fpu\_interrupt\_disable is shown as below:

**Table 3-635. Function syscfg\_fpu\_interrupt\_disable**

<b>Function name</b>	syscfg_fpu_interrupt_disable
<b>Function prototype</b>	void syscfg_fpu_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	disable FPU interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	FPU interrupt
<i>SYSCFG_FPUINT_INV ALID_OPERATION</i>	invalid operation interrupt
<i>SYSCFG_FPUINT_DIV 0</i>	divide-by-zero interrupt
<i>SYSCFG_FPUINT_UN DERFLOW</i>	overflow interrupt

<code>SYSCFG_FPUINT_OV ERFLOW</code>	underflow interrupt
<code>SYSCFG_FPUINT_INP UT_ABNORMAL</code>	input abnormal interrupt
<code>SYSCFG_FPUINT_INE XACT</code>	inexact interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable FPU inexact interrupt */
syscfg_fpu_interrupt_disable(SYSCFG_FPUINT_INEXACT);
```

### syscfg\_sram\_ecc\_interrupt\_enable

The description of syscfg\_sram\_ecc\_interrupt\_enable is shown as below:

**Table 3-636. syscfg\_sram\_ecc\_interrupt\_enable**

<b>Function name</b>	syscfg_sram_ecc_interrupt_enable
<b>Function prototype</b>	void syscfg_sram_ecc_interrupt_enable(uint32_t sram_ecc)
<b>Function descriptions</b>	enable SRAM ECC error interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sram_ecc</b>	SRAM ECC error
<code>SYSCFG_SRAM_ECCC S_SRAM_ECCMEIE</code>	SRAM multi-bit non-correction interrupt enable
<code>SYSCFG_SRAM_ECCC S_SRAM_ECCSEIE</code>	SRAM single bit correction interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SRAM ECC error interrupt */
syscfg_sram_ecc_interrupt_enable(SYSCFG_SRAM_ECCCS_SRAM_ECCMEIE);
```

### syscfg\_sram\_ecc\_interrupt\_disable

The description of syscfg\_sram\_ecc\_interrupt\_disable is shown as below:

**Table 3-637. syscfg\_sram\_ecc\_interrupt\_disable**

<b>Function name</b>	syscfg_sram_ecc_interrupt_disable
<b>Function prototype</b>	void syscfg_sram_ecc_interrupt_disable(uint32_t sram_ecc)
<b>Function descriptions</b>	disable SRAM ECC error interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sram_ecc</b>	SRAM ECC error
SYSCFG_SRAM_ECCCS_SRAM_ECCMEIE	SRAM multi-bit non-correction interrupt enable
SYSCFG_SRAM_ECCCS_SRAM_ECCSEIE	SRAM single bit correction interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SRAM ECC error interrupt */
syscfg_sram_ecc_interrupt_disable(SYSCFG_SRAM_ECCCS_SRAM_ECCMEIE);
```

### syscfg\_bus\_timeout\_flag\_get

The description of syscfg\_bus\_timeout\_flag\_get is shown as below:

**Table 3-638. syscfg\_bus\_timeout\_flag\_get**

<b>Function name</b>	syscfg_bus_timeout_flag_get
<b>Function prototype</b>	FlagStatus syscfg_bus_timeout_flag_get(uint32_t bustoflag)
<b>Function descriptions</b>	get bus timeout flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>bustoflag</b>	bus timeout flag
SYSCFG_BUSTOFLAG_CPU_CBUS	CPU Cbus timeout flag
SYSCFG_BUSTOFLAG_CPU_SBUS	CPU Sbus timeout flag
SYSCFG_BUSTOFLAG_DMA0	DMA0 bus timeout flag
SYSCFG_BUSTOFLAG_DMA1	DMA1 bus timeout flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	



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

Example:

```
/* get bus timeout flag */
```

```
syscfg_bus_timeout_flag_get(CPU Cbus timeout flag);
```

### syscfg\_bus\_timeout\_flag\_clear

The description of syscfg\_bus\_timeout\_flag\_clear is shown as below:

**Table 3-639. syscfg\_bus\_timeout\_flag\_clear**

<b>Function name</b>	syscfg_bus_timeout_flag_clear
<b>Function prototype</b>	void syscfg_bus_timeout_flag_clear(uint32_t bustoflag)
<b>Function descriptions</b>	clear bus timeout flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>bustoflag</b>	bus timeout flag
SYSCFG_BUSTOSTAT_CPUCBUSTOF	CPU Cbus timeout flag
SYSCFG_BUSTOSTAT_CPUSBUSTOF	CPU Sbus timeout flag
SYSCFG_BUSTOSTAT_DMA0BUSTOF	DMA0 bus timeout flag
SYSCFG_BUSTOSTAT_DMA1BUSTOF	DMA1 bus timeout flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear bus timeout flag */
```

```
syscfg_bus_timeout_flag_clear(SYSCFG_BUSTOSTAT_CPUCBUSTOF);
```

### syscfg\_sram\_ecc\_flag\_get

The description of syscfg\_sram\_ecc\_flag\_get is shown as below:

**Table 3-640. syscfg\_sram\_ecc\_flag\_get**

<b>Function name</b>	syscfg_sram_ecc_flag_get
<b>Function prototype</b>	FlagStatus syscfg_sram_ecc_flag_get(uint32_t ecc_flag)
<b>Function descriptions</b>	get SRAM ECC error flag
<b>Precondition</b>	-

The called functions	-
Input parameter{in}	
<b>ecc_flag</b>	SRAM ECC error flag
<i>SYSCFG_SRAM_ECCS</i> <i>TAT_SRAM_ECCMEIF</i>	SRAM non-correction event detected
<i>SYSCFG_SRAM_ECCS</i> <i>TAT_SRAM_ECCSEIF</i>	SRAM single bit correction event detected
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get SRAM ECC error flag */
syscfg_sram_ecc_flag_get(SYSCFG_SRAM_ECCSTAT_SRAM_ECCMEIF);
```

### syscfg\_sram\_ecc\_flag\_clear

The description of syscfg\_sram\_ecc\_flag\_clear is shown as below:

**Table 3-641. syscfg\_sram\_ecc\_flag\_clear**

<b>Function name</b>	syscfg_sram_ecc_flag_clear
<b>Function prototype</b>	void syscfg_sram_ecc_flag_clear(uint32_t ecc_flag)
<b>Function descriptions</b>	clear SRAM ECC error flag
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>ecc_flag</b>	SRAM ECC error flag
<i>SYSCFG_SRAM_ECCS</i> <i>TAT_SRAM_ECCMEIF</i>	SRAM non-correction event detected
<i>SYSCFG_SRAM_ECCS</i> <i>TAT_SRAM_ECCSEIF</i>	SRAM single bit correction event detected
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear SRAM ECC error flag */
syscfg_sram_ecc_flag_clear(SYSCFG_SRAM_ECCSTAT_SRAM_ECCMEIF);
```

## 3.24. SPI

The SPI/I2S module can communicate with external devices using the SPI protocol or the I2S audio protocol. The SPI/I2S registers are listed in chapter [3.24.1](#), the SPI/I2S firmware functions are introduced in chapter [3.24.2](#).

### 3.24.1. Descriptions of Peripheral registers

SPI/I2S registers are listed in the table shown as below:

**Table 3-642. SPI/I2S Registers**

Registers	Descriptions
SPI_CTL0	SPI control register 0
SPI_CTL1	SPI control register 1
SPI_STAT	SPI status register
SPI_DATA	SPI data register
SPI_CRCPOLY	SPI CRC polynomial register
SPI_RCRC	SPI receive CRC register
SPI_TCRC	SPI transmit CRC register
SPI_I2SCTL	SPI/I2S control register
SPI_I2SPSC	SPI/I2S clock prescaler register
SPI_QCTL	Quad-SPI mode control register

### 3.24.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

**Table 3-643. 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 parameters
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

Function name	Function description
spi_dma_disable	disable SPI DMA send or receive
spi_i2s_data_frame_format_config	configure SPI data frame format
spi_i2s_data_transmit	SPI transmit data
spi_i2s_data_receive	SPI receive data
spi_bidirectional_transfer_config	configure SPI bidirectional transfer direction
spi_i2s_format_error_clear	clear SPI/I2S format error flag status
spi_crc_polynomial_set	set SPI CRC polynomial
spi_crc_polynomial_get	get SPI CRC polynomial
spi_crc_on	turn on SPI CRC function
spi_crc_off	turn off SPI CRC function
spi_crc_next	SPI next data is CRC value
spi_crc_get	get SPI CRC send value or receive value
spi_crc_error_clear	clear SPI CRC error flag status
spi_ti_mode_enable	enable SPI TI mode
spi_ti_mode_disable	disable SPI TI mode
spi_nssp_mode_enable	enable SPI NSS pulse mode
spi_nssp_mode_disable	disable SPI NSS pulse mode
spi_quad_enable	enable quad wire SPI
spi_quad_disable	disable quad wire SPI
spi_quad_write_enable	enable quad wire SPI write
spi_quad_read_enable	enable quad wire SPI read
spi_i2s_flag_get	get SPI and I2S flag status
spi_i2s_interrupt_enable	enable SPI and I2S interrupt
spi_i2s_interrupt_disable	disable SPI and I2S interrupt
spi_i2s_interrupt_flag_get	get SPI and I2S interrupt status

## Structure spi\_parameter\_struct

Table 3-644. spi\_parameter\_struct

Member name	Function description
device_mode	SPI master or slave (SPI_MASTER, SPI_SLAVE)
trans_mode	SPI transfer type (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	SPI frame size (SPI_FRAME_SIZE_16BIT, SPI_FRAME_SIZE_8BIT)
nss	SPI NSS control by hardware or software (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	SPI big endian or little endian (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	SPI clock phase and polarity (SPI_CLOCK_PL_LOW_PHASE_1EDGE, SPI_CLOCK_PL_HIGH_PHASE_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))

## Enum spi\_interrupt\_flag\_enum

**Table 3-645. spi\_interrupt\_flag\_enum**

Member name	Function description
SPI_I2S_INT_FLAG_TBE	Transmit buffer empty interrupt flag
SPI_I2S_INT_FLAG_RBNE	Receive buffer not empty interrupt flag
SPI_I2S_INT_FLAG_RXOVERR	RX overrun error interrupt flag
SPI_INT_FLAG_CONFERR	Config error interrupt flag
SPI_INT_FLAG_CRCERR	CRC error interrupt flag
I2S_INT_FLAG_TXURERR	TX underrun error interrupt flag
SPI_I2S_INT_FLAG_FERR	Format error interrupt flag

## spi\_i2s\_deinit

The description of spi\_i2s\_deinit is shown as below:

**Table 3-646. Function spi\_i2s\_deinit**

Function name	spi_i2s_deinit
Function prototype	void spi_i2s_deinit(uint32_t spi_periph);
Function descriptions	reset SPI and I2S
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
spi_periph	SPI/I2S peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset SPI0 */
spi_i2s_deinit(SPI0);
```

## spi\_struct\_para\_init

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

**Table 3-647. 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 3-644. 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 3-648. 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 parameters
<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 structure, the structure members can refer to members of the structure <a href="#">Table 3-644. spi_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* initialize SPI0 */

spi_parameter_struct spi_init_struct;

spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode     = SPI_MASTER;
spi_init_struct.frame_size      = SPI_FRAME_SIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CLOCK_POLARITY_HIGH_PHASE_2EDGE;
spi_init_struct.nss              = SPI_NSS_SOFT;
spi_init_struct.prescale        = SPI_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 3-649. Function spi\_enable**

<b>Function name</b>	spi_enable
<b>Function prototype</b>	void spi_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* enable SPI0 */

spi_enable(SPI0);

```

## spi\_disable

The description of spi\_disable is shown as below:

**Table 3-650. Function spi\_disable**

<b>Function name</b>	spi_disable
<b>Function prototype</b>	void spi_disable(uint32_t spi_periph);

<b>Function descriptions</b>	disable SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 */
spi_disable(SPI0);
```

## i2s\_init

The description of i2s\_init is shown as below:

**Table 3-651. Function i2s\_init**

<b>Function name</b>	i2s_init
<b>Function prototype</b>	void i2s_init(uint32_t spi_periph, uint32_t i2s_mode, uint32_t i2s_standard, uint32_t i2s_ckpl);
<b>Function descriptions</b>	initialize I2S pparameters
<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_SLAVRX</i>	I2S slave receive mode
<i>I2S_MODE_MASTERTX</i>	I2S master transmit mode
<i>I2S_MODE_MASTERRX</i>	I2S master receive mode
<b>Input parameter{in}</b>	
<b>i2s_standard</b>	I2S standard
<i>I2S_STD_PHILIPS</i>	I2S philips standard
<i>I2S_STD_MSB</i>	I2S MSB standard
<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCMSHORT</i>	I2S PCM short standard
<i>I2S_STD_PCMLONG</i>	I2S PCM long standard



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

Example:

```
/* initialize I2S1 */
```

```
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILIPS, I2S_CKPL_LOW);
```

### i2s\_psc\_config

The description of i2s\_psc\_config is shown as below:

**Table 3-652. 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
Input parameter{in}	
<b>spi_periph</b>	I2S peripheral
<i>SPIx</i>	x=1,2
Input parameter{in}	
<b>i2s_audiosample</b>	I2S audio sample rate
<i>I2S_AUDIOSAMPLE_8K</i>	audio sample rate is 8KHz
<i>I2S_AUDIOSAMPLE_11K</i>	audio sample rate is 11KHz
<i>I2S_AUDIOSAMPLE_16K</i>	audio sample rate is 16KHz
<i>I2S_AUDIOSAMPLE_22K</i>	audio sample rate is 22KHz
<i>I2S_AUDIOSAMPLE_32K</i>	audio sample rate is 32KHz
<i>I2S_AUDIOSAMPLE_44K</i>	audio sample rate is 44KHz
<i>I2S_AUDIOSAMPLE_48K</i>	audio sample rate is 48KHz
<i>I2S_AUDIOSAMPLE_96K</i>	audio sample rate is 96KHz

6K	
I2S_AUDIOSAMPLE_1 92K	audio sample rate is 192KHz
<b>Input parameter{in}</b>	
i2s_frameformat	I2S data length and channel length
I2S_FRAMEFORMAT_ DT16B_CH16B	I2S data length is 16 bit and channel length is 16 bit
I2S_FRAMEFORMAT_ DT16B_CH32B	I2S data length is 16 bit and channel length is 32 bit
I2S_FRAMEFORMAT_ DT24B_CH32B	I2S data length is 24 bit and channel length is 32 bit
I2S_FRAMEFORMAT_ DT32B_CH32B	I2S data length is 32 bit and channel length is 32 bit
<b>Input parameter{in}</b>	
i2s_mckout	I2S master clock output
I2S_MCKOUT_ENABL E	I2S master clock output enable
I2S_MCKOUT_DISABL E	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 3-653. 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>	
spi_periph	I2S peripheral
SPIx	x=1,2
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* enable I2S1*/
i2s_enable(SPI1);
```

### i2s\_disable

The description of i2s\_disable is shown as below:

**Table 3-654. Function i2s\_disable**

Function name	i2s_disable
Function prototype	void i2s_disable(uint32_t spi_periph);
Function descriptions	disable I2S
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	I2S peripheral
SPIx	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2S1*/
i2s_disable(SPI1);
```

### spi\_nss\_output\_enable

The description of spi\_nss\_output\_enable is shown as below:

**Table 3-655. Function spi\_nss\_output\_enable**

Function name	spi_nss_output_enable
Function prototype	void spi_nss_output_enable(uint32_t spi_periph);
Function descriptions	enable SPI NSS output
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* enable SPI0 NSS output */
spi_nss_output_enable(SPI0);
```

### spi\_nss\_output\_disable

The description of spi\_nss\_output\_disable is shown as below:

**Table 3-656. 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 3-657. 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 3-658. 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 3-659. Function spi\_dma\_enable**

Function name	spi_dma_enable
Function prototype	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
Function descriptions	enable SPI DMA send or receive
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Input parameter{in}	
dma	SPI DMA mode

<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA
<i>SPI_DMA_RECEIVE</i>	SPI receive data use DMA
<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 3-660. Function spi\_dma\_disable**

<b>Function name</b>	spi_dma_disable
<b>Function prototype</b>	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
<b>Function descriptions</b>	disable SPI DMA 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>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 3-661. 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 data frame format
<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>Input parameter{in}</b>	
<b>frame_format</b>	SPI frame size
<b>SPI_FRAME_SIZE_16BIT</b>	SPI frame size is 16 bits
<b>SPI_FRAME_SIZE_8BIT</b>	SPI frame size is 8 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SPI1/I2S1 data frame format size is 16 bits */
```

```
spi_i2s_data_frame_format_config(SPI1, SPI_FRAME_SIZE_16BIT);
```

## spi\_i2s\_data\_transmit

The description of spi\_i2s\_data\_transmit is shown as below:

**Table 3-662. Function spi\_i2s\_data\_transmit**

<b>Function name</b>	spi_i2s_data_transmit
<b>Function prototype</b>	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
<b>Function descriptions</b>	SPI transmit data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1,2
<b>Input parameter{in}</b>	
<b>data</b>	16-bit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 transmit data */
```

```
uint16_t spi0_send_array[] = {0x5050,0xA0A0};

spi_i2s_data_transmit(SPI0, spi0_send_array[0]);
```

### spi\_i2s\_data\_receive

The description of spi\_i2s\_data\_receive is shown as below:

**Table 3-663. Function spi\_i2s\_data\_receive**

<b>Function name</b>	spi_i2s_data_receive
<b>Function prototype</b>	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
<b>Function descriptions</b>	SPI receive data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<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 3-664. Function spi\_bidirectional\_transfer\_config**

<b>Function name</b>	spi_bidirectional_transfer_config
<b>Function prototype</b>	void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);
<b>Function descriptions</b>	configure SPI bidirectional transfer direction
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>transfer_direction</b>	SPI transfer direction
<i>SPI_BIDIRECTIONAL_TRANSMIT</i>	SPI work in transmit-only mode



<i>SPI_BIDIRECTIONAL_RECEIVE</i>	SPI work in receive-only mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

### spi\_crc\_polynomial\_set

The description of spi\_crc\_polynomial\_set is shown as below:

**Table 3-665. Function spi\_crc\_polynomial\_set**

<b>Function name</b>	spi_crc_polynomial_set
<b>Function prototype</b>	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);
<b>Function descriptions</b>	set SPI CRC polynomial
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>crc_poly</b>	CRC polynomial value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set SPI0 CRC polynomial */
```

```
spi_crc_polynomial_set(SPI0,CRC_VALUE);
```

### spi\_crc\_polynomial\_get

The description of spi\_crc\_polynomial\_get is shown as below:

**Table 3-666. Function spi\_crc\_polynomial\_get**

<b>Function name</b>	spi_crc_polynomial_get
<b>Function prototype</b>	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
<b>Function descriptions</b>	get SPI CRC polynomial
<b>Precondition</b>	-

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 3-667. Function spi\_crc\_on**

Function name	spi_crc_on
Function prototype	void spi_crc_on(uint32_t spi_periph);
Function descriptions	turn on CRC function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn on SPI0 CRC function */
spi_crc_on(SPI0);
```

## spi\_crc\_off

The description of spi\_crc\_off is shown as below:

**Table 3-668. Function spi\_crc\_off**

Function name	spi_crc_off
Function prototype	void spi_crc_off(uint32_t spi_periph);
Function descriptions	turn off CRC function

<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 off SPI0 CRC function */
```

```
spi_crc_off(SPI0);
```

### spi\_crc\_next

The description of spi\_crc\_next is shown as below:

**Table 3-669. Function spi\_crc\_next**

<b>Function name</b>	spi_crc_next
<b>Function prototype</b>	void spi_crc_next(uint32_t spi_periph);
<b>Function descriptions</b>	SPI next data is CRC value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 next data is CRC value */
```

```
spi_crc_next(SPI0);
```

### spi\_crc\_get

The description of spi\_crc\_get is shown as below:

**Table 3-670. Function spi\_crc\_get**

<b>Function name</b>	spi_crc_get
<b>Function prototype</b>	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc);
<b>Function descriptions</b>	get SPI CRC send value or receive value

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>crc</b>	SPI crc value
<i>SPI_CRC_TX</i>	get transmit crc value
<i>SPI_CRC_RX</i>	get receive crc value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	16-bit CRC value (0-0xFFFF)

Example:

```
/* get SPI0 CRC send value */
uint16_t crc_val;
crc_val = spi_crc_get(SPI0, SPI_CRC_TX);
```

### spi\_crc\_error\_clear

The description of spi\_crc\_error\_clear is shown as below:

**Table 3-671. 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);
```

## spi\_ti\_mode\_enable

The description of spi\_ti\_mode\_enable is shown as below:

**Table 3-672. Function spi\_ti\_mode\_enable**

<b>Function name</b>	spi_ti_mode_enable
<b>Function prototype</b>	void spi_ti_mode_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI TI mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<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 3-673. 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 3-674. 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 3-675. 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);
```

## spi\_quad\_enable

The description of spi\_quad\_enable is shown as below:

**Table 3-676. Function spi\_quad\_enable**

<b>Function name</b>	spi_quad_enable
<b>Function prototype</b>	void spi_quad_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable quad wire SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 quad wire mode */
spi_quad_enable(SPI0);
```

## spi\_quad\_disable

The description of spi\_quad\_disable is shown as below:

**Table 3-677. Function spi\_quad\_disable**

<b>Function name</b>	spi_quad_disable
<b>Function prototype</b>	spi_quad_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable quad wire SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 quad wire mode */
spi_quad_disable(SPI0);
```

## spi\_quad\_write\_enable

The description of spi\_quad\_write\_enable is shown as below:

**Table 3-678. Function spi\_quad\_write\_enable**

<b>Function name</b>	spi_quad_write_enable
<b>Function prototype</b>	void spi_quad_write_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable quad wire SPI write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 quad wire write */
spi_quad_write_enable(SPI0);
```

## spi\_quad\_read\_enable

The description of spi\_quad\_read\_enable is shown as below:

**Table 3-679. Function spi\_quad\_read\_enable**

<b>Function name</b>	spi_quad_read_enable
<b>Function prototype</b>	void spi_quad_read_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable quad wire SPI read
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 quad wire read */
spi_quad_read_enable(SPI0);
```



## spi\_i2s\_flag\_get

The description of spi\_i2s\_flag\_get is shown as below:

**Table 3-680. Function spi\_i2s\_flag\_get**

<b>Function name</b>	spi_i2s_flag_get
<b>Function prototype</b>	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);
<b>Function descriptions</b>	get SPI and I2S flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2
<b>Input parameter{in}</b>	
<b>flag</b>	SPI/I2S flag status
<i>SPI_FLAG_TBE</i>	transmit buffer empty flag
<i>SPI_FLAG_RBNE</i>	receive buffer not empty flag
<i>SPI_FLAG_TRANS</i>	transmit on-going flag
<i>SPI_I2S_INT_FLAG_RXORERR</i>	receive overrun error flag
<i>SPI_FLAG_CONFERR</i>	mode config error flag
<i>SPI_FLAG_CRCERR</i>	CRC error flag
<i>SPI_FLAG_FREE</i>	format error flag
<i>I2S_FLAG_TBE</i>	transmit buffer empty flag
<i>I2S_FLAG_RBNE</i>	receive buffer not empty flag
<i>I2S_FLAG_TRANS</i>	transmit on-going flag
<i>I2S_FLAG_RXORERR</i>	overrun error flag
<i>I2S_FLAG_TXURERR</i>	underrun error flag
<i>I2S_FLAG_CH</i>	channel side flag
<i>I2S_FLAG_FERR</i>	format error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty flag status */
while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

## spi\_i2s\_interrupt\_enable

The description of spi\_i2s\_interrupt\_enable is shown as below:

Table 3-681. 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 3-682. 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>	

-	-
Return value	
-	-

Example:

```
/* disable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

### spi\_i2s\_interrupt\_flag\_get

The description of spi\_i2s\_interrupt\_flag\_get is shown as below:

**Table 3-683. 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, spi_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	get SPI and I2S interrupt 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>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt flag status
<i>SPI_I2S_INT_FLAG_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_FLAG_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_FLAG_RXORERR</i>	overrun interrupt
<i>SPI_INT_FLAG_CONFIGERR</i>	config error interrupt
<i>SPI_INT_FLAG_CRCERR</i>	CRC error interrupt
<i>I2S_INT_FLAG_TXURERR</i>	underrun error interrupt
<i>SPI_I2S_INT_FLAG_FORMATERR</i>	format error interrupt
<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++]);

}

```

## 3.25. TIMER

The timers have a counter that can be used as an unsigned counter and supports both input capture and output compare. Timers (TIMERx) are divided into four sorts: advanced timer (TIMERx, x=0, 7), general level0 timer (TIMERx, x=1~4), general level3 timer (TIMERx, x=15, 16) and basic timer (TIMERx, x=5, 6). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.25.1](#), the TIMER firmware functions are introduced in chapter [3.25.2](#).

### 3.25.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

**Table 3-684. 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_CREP0	Counter repetition register 0
TIMER_CH0CV	Channel 0 capture/compare value register
TIMER_CH1CV	Channel 1 capture/compare value register
TIMER_CH2CV	Channel 2 capture/compare value register
TIMER_CH3CV	Channel 3 capture/compare value register
TIMER_CCHP0	Channel complementary protection register 0
TIMER_MCHCTL0	TIMER multi mode channel control register 0
TIMER_MCHCTL2	TIMER multi mode channel control register 2

Registers	Descriptions
TIMER_MCH0CV	TIMER multi mode channel 0 capture or compare value register
TIMER_CH0COMV_A DD	TIMER channel 0 additional compare value register
TIMER_CH1COMV_A DD	TIMER channel 1 additional compare value register
TIMER_CH2COMV_A DD	TIMER channel 2 additional compare value register
TIMER_CH3COMV_A DD	TIMER channel 3 additional compare value register
TIMER_CTL2	TIMER control register 2
TIMER_AFCTL0	TIMER alternate function control register 0
TIMER_CCHP1	Channel complementary protection register 1
TIMER_WDGP	TIMER watchdog counter period register
TIMER_CINITCTL	TIMER counter initial control register
TIMER_CINITV	TIMER counter initial value register
TIMER_CHBRKCTL	TIMER channel break control register
TIMER_CHBRKPER	TIMER channel break period register
TIMER_CHBRKINTF	TIMER channel break interrupt flag register
TIMER_DMCFG	DMA configuration register
TIMER_DMATB	DMA transfer buffer register
TIMER_CFG	Configuration register

### 3.25.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-685. 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

Function name	Function description
timer_runtime_repetition_value_read	configure TIMER runtime repetition register value
timer_autoreload_value_config	configure TIMER autoreload register value
timer_autoreload_value_read	read TIMER autoreload register value
timer_counter_value_config	configure TIMER counter register value
timer_counter_read	read TIMER counter value
timer_prescaler_read	read TIMER prescaler value
timer_single_pulse_mode_config	configure TIMER single pulse mode
timer_update_source_config	configure TIMER update source
timer_dma_enable	enable the TIMER DMA
timer_dma_disable	disable the TIMER DMA
timer_channel_dma_request_source_select	channel DMA request source selection
timer_dma_transfer_config	configure the TIMER DMA transfer
timer_event_software_generate	software generate events
timer_break_struct_para_init	initialize the parameters of TIMER break parameter struct with the default values
timer_break_config	configure TIMER break function
timer_break_enable	enable TIMER break function
timer_break_disable	disable TIMER break function
timer_automatic_output_enable	enable TIMER output automatic function
timer_automatic_output_disable	disable TIMER output automatic function
timer_primary_output_config	configure TIMER primary output function
timer_channel_control_shadow_config	configure channel commutation control shadow register
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_compare_fast_config	configure TIMER channel output fast function
timer_channel_output_clear_config	configure TIMER channel output clear function
timer_channel_output_polarity_config	configure TIMER channel output polarity
timer_channel_complementary_output_polarity_config	configure TIMER channel complementary output polarity
timer_channel_output_state_config	configure TIMER channel enable state

Function name	Function description
timer_channel_complementary_output_state_config	configure TIMER channel complementary output enable state
timer_channel_composite_asymmetric_pwm_level_config	configure TIMER channel period point match moment OxCPRE level in the composite PWM or asymmetric PWM mode
timer_channel_output_clear_invalid_selection	output clear invalid event selection
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_multi_mode_channel_output_parameter_struct_init	initialize TIMER multi mode channel output parameter struct
timer_multi_mode_channel_output_config	configure TIMER multi mode channel output function
timer_multi_mode_channel_mode_config	multi mode channel mode select
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_non_quadrature_decoder_mode_config	configure TIMER non-quadrature decoder mode
timer_internal_clock_config	configure TIMER internal clock mode
timer_internal_trigger_as_external_clock_config	configure TIMER the internal trigger as external clock input
timer_external_trigger_as_external_clock_config	configure TIMER the external trigger as external clock input
timer_external_clock_mode0_config	configure TIMER the external clock mode 0
timer_external_clock_mode1_config	configure TIMER the external clock mode 1
timer_external_clock_mode1_disable	disable TIMER the external clock mode 1
timer_write_chxval_register_config	configure TIMER write CHxVAL register selection

Function name	Function description
timer_output_value_selection_config	configure TIMER output value selection
timer_commutation_control_shadow_register_config	configure commutation control shadow register update selection
timer_output_match_pulse_select	configure TIMER output match pulse selection
timer_channel_composite_pwm_mode_config	configure the TIMER composite PWM mode
timer_channel_composite_pwm_mode_output_pulse_value_config	configure the TIMER composite PWM mode output pulse value
timer_channel_asymmetric_pwm_pulse_value_config	configure the TIMER asymmetric PWM mode output pulse value
timer_channel_additional_compare_value_config	configure TIMER channel additional compare value
timer_channel_additional_output_shadow_config	configure TIMER channel additional output shadow function
timer_channel_additional_output_update_select	select TIMER channel additional output register update source
timer_channel_additional_compare_value_read	read TIMER channel additional compare value
timer_break_external_source_config	configure TIMER break external source
timer_break_external_polarity_config	configure TIMER break polarity
timer_dead_time_falling_edge_config	configure the TIMER falling edge dead time
timer_dead_time_different_config	configure the TIMER dead time different function
timer_channel_break_control_config	configure the TIMER channel break function
timer_channel_dead_time_config	configure the TIMER channel dead time function
timer_channel_break_config	configure the TIMER channel break
timer_channel_break_external_status_config	configure TIMER channel break external input enable
timer_channel_break_external_polarity_config	configure TIMER channel break external input polarity
timer_channel_primary_output_config	configure TIMER channel primary output function
timer_channel_break_period_config	configure TIMER channel break period value
timer_watchdog_value_config	configure quadrature decoder signal disconnection detection watchdog value
timer_watchdog_value_read	read quadrature decoder signal disconnection detection watchdog value
timer_decoder_disconnection_detection_config	configure quadrature decoder signal disconnection detection function



Function name	Function description
timer_decoder_jump_detection_config	configure decoder signal jump detection function
timer_counter_initial_register_config	configure counter initial value register
timer_counter_initial_config	configure counter initial value and direction
timer_synchronization_event_generate	generate soft synchronization event
timer_flag_get	get TIMER flags
timer_flag_clear	clear TIMER flags
timer_interrupt_enable	enable the TIMER interrupt
timer_interrupt_disable	disable the TIMER interrupt
timer_interrupt_flag_get	get timer interrupt flags
timer_interrupt_flag_clear	clear TIMER interrupt flags

### Structure timer\_parameter\_struct

**Table 3-686. 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(0~0xFFFF(TIMERx(x=0,2~7,15,16)), 0~0xFFFFFFFF(TIMERx(x=1)))
clockdivision	clock division value(TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)
repetitioncounter	the counter repetition value(0~0xFF)

### Structure timer\_break\_parameter\_struct

**Table 3-687. Structure timer\_break\_parameter\_struct**

Member name	Function description
runoffstate	run mode off-state(TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE)
ideloffstate	idle mode off-state(TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE)
deadtime	dead time(0~255)
outputautostate	output automatic enable (TIMER_OUTAUTO_ENABLE, TIMER_OUTAUTO_DISABLE)
protectmode	complementary register protect control(TIMER_CCHP0_PROT_OFF, TIMER_CCHP0_PROT_0, TIMER_CCHP0_PROT_1, TIMER_CCHP0_PROT_2)
break0state	BREAK0 input enable (TIMER_BREAK0_ENABLE,

Member name	Function description
	TIMER_BREAK0_DISABLE)
break0filter	BREAK0 input filter(0~15)
break0polarity	BREAK0 input polarity(TIMER_BREAK0_POLARITY_LOW, TIMER_BREAK0_POLARITY_HIGH)

### Structure timer\_oc\_parameter\_struct

**Table 3-688. 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\_omc\_parameter\_struct

**Table 3-689. Structure timer\_omc\_parameter\_struct**

Member name	Function description
outputmode	multi mode channel output mode selection(TIMER_MCH_MODE_INDEPENDENTLY, TIMER_MCH_MODE_COMPLEMENTARY)
outputstate	multi mode channel output state(TIMER_MCCX_ENABLE, TIMER_MCCX_DISABLE)
ocpolarity	multi mode channel output polarity(TIMER_OMC_POLARITY_HIGH, TIMER_OMC_POLARITY_LOW)

### Structure timer\_ic\_parameter\_struct

**Table 3-690. Structure timer\_ic\_parameter\_struct**

Member name	Function description
icpolarity	channel input polarity(TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)
icselection	channel input mode selection(TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS)
icprescaler	channel input capture prescaler(TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)
icfilter	channel input capture filter control(0~15)

## timer\_deinit

The description of timer\_deinit is shown as below:

**Table 3-691. 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=0~7, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset TIMER0 */
timer_deinit(TIMER0);
```

## timer\_struct\_para\_init

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

**Table 3-692. 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="#">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 3-693. 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=0~7, 15, 16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>initpara</b>	TIMER init parameter struct, the structure members can refer to <a href="#">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 3-694. 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=0~7, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 */
timer_enable(TIMER0);
```

### timer\_disable

The description of timer\_disable is shown as below:

**Table 3-695. 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=0~7, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 */
timer_disable(TIMER0);
```

### timer\_auto\_reload\_shadow\_enable

The description of timer\_auto\_reload\_shadow\_enable is shown as below:

**Table 3-696. 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=0~7, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable(TIMER0);
```

### timer\_auto\_reload\_shadow\_disable

The description of timer\_auto\_reload\_shadow\_disable is shown as below:

**Table 3-697. 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=0~7, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable(TIMER0);
```

### timer\_update\_event\_enable

The description of timer\_update\_event\_enable is shown as below:

**Table 3-698. 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=0~7, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 the update event */
timer_update_event_enable (TIMER0);
```

### timer\_update\_event\_disable

The description of timer\_update\_event\_disable is shown as below:

**Table 3-699. Function timer\_update\_event\_disable**

<b>Function name</b>	timer_update_event_disable
<b>Function prototype</b>	void timer_update_event_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable the update event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~7, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 the update event */
timer_update_event_disable (TIMER0);
```

### timer\_counter\_alignment

The description of timer\_counter\_alignment is shown as below:

**Table 3-700. 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> =0~4,7,15,16)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>aligned</b>	alignment mode
<i>TIMER_COUNTER_EDGE</i>	edge-aligned mode
<i>TIMER_COUNTER_CENTERTDOWN</i>	center-aligned and counting down assert mode
<i>TIMER_COUNTER_CENTERTUP</i>	center-aligned and counting up assert mode
<i>TIMER_COUNTER_CENTERTBOTH</i>	center-aligned and counting up/down assert mode
<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 3-701. 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 TIMERx 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</i> ( <i>x</i> =0~4,7,15,16)	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 3-702. 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 TIMERx 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=0~4,7,15,16)</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 3-703. Function timer\_prescaler\_config**

<b>Function name</b>	timer_prescaler_config
<b>Function prototype</b>	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint32_t pscreload);
<b>Function descriptions</b>	configure TIMER prescaler
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~7,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>prescaler</b>	prescaler value (0~0xFFFF)
<b>Input parameter{in}</b>	
<b>pscreload</b>	prescaler reload mode
<i>TIMER_PSC_RELOAD</i>	the prescaler is loaded right now

<code>_NOW</code>	
<code>TIMER_PSC_RELOAD</code> <code>_UPDATE</code>	the prescaler is loaded at the next update event
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 prescaler */
timer_prescaler_config(TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

### timer\_repetition\_value\_config

The description of timer\_repetition\_value\_config is shown as below:

**Table 3-704. Function timer\_repetition\_value\_config**

<b>Function name</b>	timer_repetition_value_config
<b>Function prototype</b>	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
<b>Function descriptions</b>	configure TIMER repetition register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<code>TIMERx(x=0,7,15,16)</code>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>repetition</b>	the counter repetition value (0~0xFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 repetition register 0 value */
timer_repetition_value_config(TIMER0, 98);
```

### timer\_runtime\_repetition\_value\_read

The description of timer\_runtime\_repetition\_value\_read is shown as below:

**Table 3-705. Function timer\_runtime\_repetition\_value\_read**

<b>Function name</b>	timer_runtime_repetition_value_read
<b>Function prototype</b>	uint32_t timer_runtime_repetition_value_read(uint32_t timer_periph);

<b>Function descriptions</b>	read TIMER runtime repetition register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	counter repetition value in TIMER_CREP0 register (0~0xFF)

Example:

```
/* read TIMER0 runtime repetition register value */
uint32_t i = 0;
i = timer_runtime_repetition_value_read(TIMER0);
```

### timer\_autoreload\_value\_config

The description of timer\_autoreload\_value\_config is shown as below:

**Table 3-706. 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, uint64_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=0~7,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>autoreload</b>	the counter auto-reload value 0~0xFFFF:TIMERx(x=0,2,3,4,7,15,16) 0~0xFFFFFFFF, TIMERx(x=1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER autoreload register value */
timer_autoreload_value_config(TIMER0, 3000);
```

## timer\_autoreload\_value\_read

The description of timer\_autoreload\_value\_read is shown as below:

**Table 3-707. Function timer\_autoreload\_value\_read**

<b>Function name</b>	timer_autoreload_value_read
<b>Function prototype</b>	uint32_t timer_autoreload_value_read(uint32_t timer_periph);
<b>Function descriptions</b>	read 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=0~7, 15, 16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	the counter auto-reload value 0~0xFFFF:TIMERx(x=0,2,3,4,7,15,16) 0~0xFFFFFFFF, TIMERx(x=1)

Example:

```
/* get TIMER autoreload register value */
```

```
uint32_t i = 0;
```

```
i=(uint32_t) timer_autoreload_value_read (TIMER0);
```

## timer\_counter\_value\_config

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

**Table 3-708. Function timer\_counter\_value\_config**

<b>Function name</b>	timer_counter_value_config
<b>Function prototype</b>	void timer_counter_value_config(uint32_t timer_periph, uint32_t counter);
<b>Function descriptions</b>	configure TIMER counter register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~7, 15, 16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>counter</b>	the counter value 0~0xFFFF:TIMERx(x=0,2,3,4,7,15,16) 0~0xFFFFFFFF, TIMERx(x=1)
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure TIMER0 counter register value */
```

```
timer_counter_value_config(TIMER0, 999);
```

### timer\_counter\_read

The description of timer\_counter\_read is shown as below:

**Table 3-709. Function timer\_counter\_read**

Function name	timer_counter_read
Function prototype	uint32_t timer_counter_read(uint32_t timer_periph);
Function descriptions	read TIMER counter value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0~7,15,16)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
uint32_t	counter value 0~0xFFFF:TIMERx(x=0,2,3,4,7,15,16) 0~0xFFFFFFFF, TIMERx(x=1)

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 3-710. Function timer\_prescaler\_read**

Function name	timer_prescaler_read
Function prototype	uint16_t timer_prescaler_read(uint32_t timer_periph);
Function descriptions	read TIMER prescaler value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral

<i>TIMERx</i> ( <i>x</i> =0~7, 15, 16)	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	prescaler register value (0~0xFFFF)

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 3-711. Function timer\_single\_pulse\_mode\_config**

<b>Function name</b>	timer_single_pulse_mode_config
<b>Function prototype</b>	void timer_single_pulse_mode_config(uint32_t timer_periph, uint32_t spmode);
<b>Function descriptions</b>	configure TIMER single pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> ( <i>x</i> =0~7, 15, 16)	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 3-712. Function timer\_update\_source\_config

Function name	timer_update_source_config
Function prototype	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
Function descriptions	configure TIMER update source
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0~7, 15, 16)	TIMER peripheral selection
Input parameter{in}	
update	update source
TIMER_UPDATE_SRC_GLOBAL	any of the following events generate an update interrupt or DMA request: – The UPG bit is set – The counter generates an overflow or underflow event – The slave mode controller generates an update event
TIMER_UPDATE_SRC_REGULAR	only counter overflow/underflow generates an update interrupt or DMA request.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER update only by counter overflow/underflow */
```

```
timer_update_source_config(TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

### timer\_dma\_enable

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

Table 3-713. Function timer\_dma\_enable

Function name	timer_dma_enable
Function prototype	void timer_dma_enable(uint32_t timer_periph, uint32_t dma);
Function descriptions	enable the TIMER DMA
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0~7, 15, 16)	please refer to the following parameters
Input parameter{in}	
dma	timer DMA source enable
TIMER_DMA_UPD	update DMA request, TIMERx(x=0~7, 15, 16)
TIMER_DMA_CH0D	channel 0 DMA request, TIMERx(x=0~4, 7, 15, 16)
TIMER_DMA_CH1D	channel 1 DMA request, TIMERx(x=0~4, 7, 15, 16)

<i>TIMER_DMA_CH2D</i>	channel 2 DMA request, <i>TIMERx</i> ( <i>x</i> =0~4,7)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA request, <i>TIMERx</i> ( <i>x</i> =0~4,7)
<i>TIMER_DMA_CMTD</i>	commutation DMA request, <i>TIMERx</i> ( <i>x</i> =0,7,15,16)
<i>TIMER_DMA_TRGD</i>	trigger DMA request, <i>TIMERx</i> ( <i>x</i> =0~4,7,15,16)
<i>TIMER_DMA_MCH0D</i>	multi mode channel 0 DMA request, <i>TIMERx</i> ( <i>x</i> =15,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER0 update DMA */
```

```
timer_dma_enable(TIMER0, TIMER_DMA_UPD);
```

### timer\_dma\_disable

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

**Table 3-714. Function timer\_dma\_disable**

<b>Function name</b>	timer_dma_disable
<b>Function prototype</b>	void timer_dma_disable (uint32_t timer_periph, uint32_t dma);
<b>Function descriptions</b>	disable the TIMER DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> ( <i>x</i> =0~7,15,16)	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma</b>	timer DMA source disable
<i>TIMER_DMA_UPD</i>	update DMA request, <i>TIMERx</i> ( <i>x</i> =0~7,15,16)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA request, <i>TIMERx</i> <i>TIMERx</i> ( <i>x</i> =0~4,7,15,16)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA request, <i>TIMERx</i> ( <i>x</i> =0~4,7,15,16)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA request, <i>TIMERx</i> ( <i>x</i> =0~4,7)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA request, <i>TIMERx</i> ( <i>x</i> =0~4,7)
<i>TIMER_DMA_CMTD</i>	commutation DMA request, <i>TIMERx</i> ( <i>x</i> =0,7,15,16)
<i>TIMER_DMA_TRGD</i>	trigger DMA request, <i>TIMERx</i> ( <i>x</i> =0~4,7,15,16)
<i>TIMER_DMA_MCH0D</i>	multi mode channel 0 DMA request, <i>TIMERx</i> ( <i>x</i> =15,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER0 update DMA */
```



```
timer_dma_disable(TIMERO, TIMER_DMA_UPD);
```

### timer\_channel\_dma\_request\_source\_select

The description of timer\_channel\_dma\_request\_source\_select is shown as below:

**Table 3-715. 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>TIMERO</i> (x=0~4,7,15,16)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>dma_request</b>	channel DMA request source selection
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	DMA request of channel n is sent when channel n 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:

```
/* TIMERO channel DMA request of channel n is sent when channel event occurs */
```

```
timer_channel_dma_request_source_select(TIMERO,
TIMER_DMAREQUEST_CHANNELEVENT);
```

### timer\_dma\_transfer\_config

The description of timer\_dma\_transfer\_config is shown as below:

**Table 3-716. 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> ( <i>x</i> =0~4,7,15,16)	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma_baseaddr</b>	DMA transfer access start address
<i>TIMER_DMACFG_DMA</i> <i>TA_CTL0</i>	DMA transfer address is <i>TIMER_CTL0</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_CTL1</i>	DMA transfer address is <i>TIMER_CTL1</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_SMCFG</i>	DMA transfer address is <i>TIMER_SMCFG</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_DMAINTEN</i>	DMA transfer address is <i>TIMER_DMAINTEN</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_INTF</i>	DMA transfer address is <i>TIMER_INTF</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_SWEVG</i>	DMA transfer address is <i>TIMER_SWEVG</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_CHCTL0</i>	DMA transfer address is <i>TIMER_CHCTL0</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_CHCTL1</i>	DMA transfer address is <i>TIMER_CHCTL1</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4)
<i>TIMER_DMACFG_DMA</i> <i>TA_CHCTL2</i>	DMA transfer address is <i>TIMER_CHCTL2</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_CNT</i>	DMA transfer address is <i>TIMER_CNT</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_PSC</i>	DMA transfer address is <i>TIMER_PSC</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_CAR</i>	DMA transfer address is <i>TIMER_CAR</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_CREP0</i>	DMA transfer address is <i>TIMER_CREP0</i> , <i>TIMERx</i> ( <i>x</i> =0,7,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH0CV</i>	DMA transfer address is <i>TIMER_CH0CV</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH1CV</i>	DMA transfer address is <i>TIMER_CH1CV</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH2CV</i>	DMA transfer address is <i>TIMER_CH2CV</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH3CV</i>	DMA transfer address is <i>TIMER_CH3CV</i> , <i>TIMERx</i> ( <i>x</i> =0,7,1,2,3,4)
<i>TIMER_DMACFG_DMA</i> <i>TA_CCHP0</i>	DMA transfer address is <i>TIMER_CCHP0</i> , <i>TIMERx</i> ( <i>x</i> =0,7,15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_MCHCTL0</i>	DMA transfer address is <i>TIMER_MCHCTL0</i> , <i>TIMERx</i> ( <i>x</i> =15,16)
<i>TIMER_DMACFG_DMA</i> <i>TA_MCHCTL2</i>	DMA transfer address is <i>TIMER_MCHCTL2</i> , <i>TIMERx</i> ( <i>x</i> =15,16)

<i>TIMER_DMACFG_DMA</i> <i>TA_MCH0CV</i>	DMA transfer address is <i>TIMER_MCH0CV</i> , <i>TIMERx(x=15,16)</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CH0COMV_ADD</i>	DMA transfer address is <i>TIMER_CH0COMV_ADD</i> , <i>TIMERx(x=0,7,15,16)</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CH1COMV_ADD</i>	DMA transfer address is <i>TIMER_CH1COMV_ADD</i> , <i>TIMERx(x=0,7,15,16)</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CH2COMV_ADD</i>	DMA transfer address is <i>TIMER_CH2COMV_ADD</i> , <i>TIMERx(x=0,7)</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CH3COMV_ADD</i>	DMA transfer address is <i>TIMER_CH3COMV_ADD</i> , <i>TIMERx(x=0,7)</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CTL2</i>	DMA transfer address is <i>TIMER_CTL2</i> , <i>TIMERx(x=0,7,1,2,3,4,15,16)</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_AFCTL0</i>	DMA transfer address is <i>TIMER_AFCTL0</i> , <i>TIMERx(x=0,7,15,16)</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CCHP1</i>	DMA transfer address is <i>TIMER_CCHP1</i> , <i>TIMERx(x=0,7)</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_WDGCNT</i>	DMA transfer address is <i>TIMER_WDGCNT</i> , <i>TIMERx(x=1~4)</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CINITCTL</i>	DMA transfer address is <i>TIMER_CINITCTL</i> , <i>TIMERx(x=0,7,15,16)</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CINITV</i>	DMA transfer address is <i>TIMER_CINITV</i> , <i>TIMERx(x=0,7,15,16)</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CHBRKCTL</i>	DMA transfer address is <i>TIMER_CHBRKCTL</i> , <i>TIMERx(x=0,7)</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CHBRKPER</i>	DMA transfer address is <i>TIMER_CHBRKPER</i> , <i>TIMERx(x=0,7)</i>
<i>TIMER_DMACFG_DMA</i> <i>TA_CHBRKINTF</i>	DMA transfer address is <i>TIMER_CHBRKINTF</i> , <i>TIMERx(x=0,7)</i>
<b>Input parameter{in}</b>	
<b>dma_lenth</b>	DMA transfer count
<i>TIMER_DMACFG_DMA</i> <i>TC_xTRANSFER</i>	( <i>x=1~38</i> ), DMA transfer <i>x</i> 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 3-717. 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, uint32_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
TIMERx(x=0~7, 15, 16)	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>event</b>	the timer software event generation sources
TIMER_EVENT_SRC_UPG	update event, TIMERx(x=0~7, 15, 16)
TIMER_EVENT_SRC_CH0G	channel 0 capture or compare event generation, TIMERx(x=0, 7, 1, 2, 3, 4, 15, 16)
TIMER_EVENT_SRC_CH1G	channel 1 capture or compare event generation, TIMERx(x=0, 7, 1, 2, 3, 4, 15, 16)
TIMER_EVENT_SRC_CH2G	channel 2 capture or compare event generation, TIMERx(x=0, 7, 1, 2, 3, 4)
TIMER_EVENT_SRC_CH3G	channel 3 capture or compare event generation, TIMERx(x=0, 7, 1, 2, 3, 4)
TIMER_EVENT_SRC_CMTG	channel commutation event generation, TIMERx(x=0, 7, 15, 16)
TIMER_EVENT_SRC_TRGG	trigger event generation, TIMERx(x=0, 7, 1, 2, 3, 4, 15, 16)
TIMER_EVENT_SRC_BREAK0	BREAK0 event generation, TIMERx(x=0, 7, 15, 16)
TIMER_EVENT_SRC_MCH0G	multi mode channel 0 capture or compare event generation, TIMERx(x=15, 16)
TIMER_EVENT_SRC_CH0COMADDG	channel 0 additional compare event generation, TIMERx(x=0, 7, 15, 16)
TIMER_EVENT_SRC_CH1COMADDG	channel 1 additional compare event generation, TIMERx(x=0, 7, 15, 16)
TIMER_EVENT_SRC_CH2COMADDG	channel 2 additional compare event generation, TIMERx(x=0, 7)
TIMER_EVENT_SRC_CH3COMADDG	channel 3 additional compare event generation, TIMERx(x=0, 7)
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* software generate update event*/
```

```
timer_event_software_generate(TIMER0, TIMER_EVENT_SRC_UPG);
```

### timer\_break\_struct\_para\_init

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

**Table 3-718. 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="#">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 3-719. Function timer\_break\_config**

Function name	timer_break_config
Function prototype	void timer_break_config(uint32_t timer_periph, timer_break_parameter_struct* breakpara);
Function descriptions	configure TIMER break function
Precondition	-
The called functions	-
Input parameter{in}	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>breakpara</b>	TIMER break parameter struct, the structure members can refer to <a href="#">Structure timer break parameter struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

timer_break_parameter_struct timer_breakpara;

timer_breakpara.runoffstate      = TIMER_ROS_STATE_DISABLE;
timer_breakpara.ideloffstate     = TIMER_IOS_STATE_DISABLE;
timer_breakpara.deadtime        = 0U;
timer_breakpara.outputautostate  = TIMER_OUTAUTO_ENABLE;
timer_breakpara.protectmode      = TIMER_CCHP0_PROT_OFF;
timer_breakpara.break0state     = TIMER_BREAK0_ENABLE;
timer_breakpara.break0filter     = 0U;
timer_breakpara.break0polarity   = TIMER_BREAK0_POLARITY_LOW;

timer_break_config(TIMERO, &timer_breakpara);

```

### timer\_break\_enable

The description of timer\_break\_enable is shown as below:

**Table 3-720. 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_CCHP0 register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 BREAK0 function*/
timer_break_enable(TIMER0);
```

### timer\_break\_disable

The description of timer\_break\_disable is shown as below:

**Table 3-721. 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_CCHP0 register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 BREAK0 function*/
timer_break_disable(TIMER0);
```

### timer\_automatic\_output\_enable

The description of timer\_automatic\_output\_enable is shown as below:

**Table 3-722. Function timer\_automatic\_output\_enable**

<b>Function name</b>	timer_automatic_output_enable
<b>Function prototype</b>	void timer_automatic_output_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable TIMER output automatic function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP0 register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* enable TIMER0 output automatic function */
```

```
timer_automatic_output_enable(TIMER0);
```

### timer\_automatic\_output\_disable

The description of timer\_automatic\_output\_disable is shown as below:

**Table 3-723. Function timer\_automatic\_output\_disable**

<b>Function name</b>	timer_automatic_output_disable
<b>Function prototype</b>	void timer_automatic_output_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER output automatic function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP0 register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 output automatic function */
```

```
timer_automatic_output_disable(TIMER0);
```

### timer\_primary\_output\_config

The description of timer\_primary\_output\_config is shown as below:

**Table 3-724. 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=0,7,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	



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

Example:

```
/* enable TIMER0 primary output function */
```

```
timer_primary_output_config(TIMER0, ENABLE);
```

### timer\_channel\_control\_shadow\_config

The description of timer\_channel\_control\_shadow\_config is shown as below:

**Table 3-725. 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>	configure channel commutation control shadow register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* channel commutation 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 3-726. 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, uint32_t ccuctl);
<b>Function descriptions</b>	configure TIMER channel 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=0,7,15,16)</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
<i>TIMER_UPDATECTL_CCUOVER</i>	the shadow registers update by when the overflow event occurs
<i>TIMER_UPDATECTL_CCUUNDER</i>	the shadow registers update by when the underflow event occurs
<i>TIMER_UPDATECTL_CCUOVERUNDER</i>	the shadow registers update by when the overflow or underflow event 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 3-727. 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* ocpara);
<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="#">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 3-728. Function timer\_channel\_output\_config**

<b>Function name</b>	timer_channel_output_config
<b>Function prototype</b>	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);
<b>Function descriptions</b>	configure TIMER channel output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~4,7,15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,7,1,2,3,4,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,7,1,2,3,4,15,16))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,7,1,2,3,4))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,7,1,2,3,4))
<b>Input parameter{in}</b>	
<b>ocpara</b>	TIMER channel output parameter struct, the structure members can refer to <a href="#">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 3-729. 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, uint32_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
TIMERx(x=0~4,7,15,16)	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
TIMER_CH_0	TIMER channel 0(TIMERx(x=0~4,7,15,16))
TIMER_CH_1	TIMER channel 1(TIMERx(x=0~4,7,15,16))
TIMER_CH_2	TIMER channel 2(TIMERx(x=0~4,7))
TIMER_CH_3	TIMER channel 3(TIMERx(x=0~4,7))
TIMER_MCH_0	TIMER multi mode channel 0(TIMERx, x=15,16)
<b>Input parameter{in}</b>	
<b>ocmode</b>	channel output compare mode
TIMER_OC_MODE_TIMING	timing mode
TIMER_OC_MODE_ACTIVE	set the channel output
TIMER_OC_MODE_INACTIVE	clear the channel output
TIMER_OC_MODE_TOGGLE	toggle on match
TIMER_OC_MODE_LOW	force low mode

<i>TIMER_OC_MODE_HIG</i> <i>H</i>	force high mode
<i>TIMER_OC_MODE_PW</i> <i>M0</i>	PWM mode 0
<i>TIMER_OC_MODE_PW</i> <i>M1</i>	PWM mode 1
<i>TIMER_OC_MODE_AS</i> <i>YMMETRIC_PWM0</i>	asymmetric PWM mode 0 (TIMERx, x=0,7)
<i>TIMER_OC_MODE_AS</i> <i>YMMETRIC_PWM1</i>	asymmetric PWM mode 1 (TIMERx, x=0,7)
<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 3-730. 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(x=0~4,7,15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0~4,7,15,16)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0~4,7,15,16)
<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)
<i>TIMER_MCH_0</i>	TIMER multi mode channel 0(TIMERx, x=15,16)
<b>Input parameter{in}</b>	
<b>pulse</b>	channel output pulse value
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output pulse value */
```

```
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

### timer\_channel\_output\_shadow\_config

The description of timer\_channel\_output\_shadow\_config is shown as below:

**Table 3-731. Function timer\_channel\_output\_shadow\_config**

Function name	timer_channel_output_shadow_config
Function prototype	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
Function descriptions	configure TIMER channel output shadow function
Precondition	-
The called functions	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~4,7,15,16)</i>	please refer to the following parameters
Input parameter{in}	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0~4,7,15,16)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0~4,7,15,16)
<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)
<i>TIMER_MCH_0</i>	TIMER multi mode channel 0(TIMERx, x=15,16)
Input parameter{in}	
<b>ocshadow</b>	channel output compare shadow
<i>TIMER_OC_SHADOW_ENABLE</i>	channel output compare shadow enable
<i>TIMER_OC_SHADOW_DISABLE</i>	channel output compare shadow disable
<i>TIMER_OMC_SHADOW_ENABLE</i>	multi mode channel output compare shadow enable
<i>TIMER_OMC_SHADOW_DISABLE</i>	multi mode channel output compare shadow disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*configure TIMER0 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config (TIMER0, TIMER_CH_0,  
TIMER_OC_SHADOW_ENABLE);
```

### timer\_channel\_output\_compare\_fast\_config

The description of timer\_channel\_output\_compare\_fast\_config is shown as below:

**Table 3-732. Function timer\_channel\_output\_compare\_fast\_config**

<b>Function name</b>	timer_channel_output_compare_fast_config
<b>Function prototype</b>	void timer_channel_output_compare_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
<b>Function descriptions</b>	configure TIMER channel output compare fast function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER periphera
<i>TIMERx(x=0~4,7,15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0~4,7,15,16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0~4,7,15,16))
<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))
<i>TIMER_MCH_0</i>	TIMER multi mode channel 0 (TIMERx(x=15,16))
<b>Input parameter{in}</b>	
<b>occlear</b>	channel output clear function
<i>TIMER_OC_FAST_ENABLE</i>	channel output compare fast function enable
<i>TIMER_OC_FAST_DISABLE</i>	channel output compare fast function disable
<i>TIMER_OMC_FAST_ENABLE</i>	multi mode channel output compare fast function enable
<i>TIMER_OMC_FAST_DISABLE</i>	multi mode channel output compare fast function disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel0 output compare fast function */
```

```
timer_channel_output_compare_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 3-733. 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
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0~4,7)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0~4,7)
<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>occlear</b>	channel output clear function
<i>TIMER_OC_CLEAR_ENABLE</i>	channel output clear function enable
<i>TIMER_OC_CLEAR_DISABLE</i>	channel output clear function disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output clear function */
```

```
timer_channel_output_clear_config (TIMER0, TIMER_CH_0,  
TIMER_OC_CLEAR_ENABLE);
```

### timer\_channel\_output\_polarity\_config

The description of timer\_channel\_output\_polarity\_config is shown as below:

**Table 3-734. Function timer\_channel\_output\_polarity\_config**

<b>Function name</b>	timer_channel_output_polarity_config
<b>Function prototype</b>	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t



	channel, uint16_t ocpolarity);
<b>Function descriptions</b>	configure TIMER channel output polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> (x=0~4,7,15,16)	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0~4,7,15,16)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0~4,7,15,16)
<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)
<i>TIMER_MCH_0</i>	TIMER multi mode channel 0(TIMERx, x=15,16)
<b>Input parameter{in}</b>	
<b>ocpolarity</b>	channel output polarity
<i>TIMER_OC_POLARITY_HIGH</i>	channel output polarity is high
<i>TIMER_OC_POLARITY_LOW</i>	channel output polarity is low
<i>TIMER_OMC_POLARITY_HIGH</i>	multi mode channel output polarity is high
<i>TIMER_OMC_POLARITY_LOW</i>	multi mode channel output polarity is low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output polarity */
```

```
timer_channel_output_polarity_config(TIMER0, TIMER_CH_0,  
TIMER_OC_POLARITY_HIGH);
```

### timer\_channel\_complementary\_output\_polarity\_config

The description of timer\_channel\_complementary\_output\_polarity\_config is shown as below:

**Table 3-735. 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</i> (x=0~4,7,15,16)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0~4,7,15,16)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0~4,7,15,16)
<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>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 3-736. 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> (x=0~4,7,15,16)	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0~4,7,15,16)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0~4,7,15,16)

<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)
<i>TIMER_MCH_0</i>	TIMER multi mode channel 0(TIMERx, x=15,16)
<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
<i>TIMER_MCCX_ENABLE</i>	multi mode channel enable
<i>TIMER_MCCX_DISABLE</i>	multi mode channel disable
<i>E</i>	
<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 3-737. 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=0,7,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0~4,7,15,16)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0~4,7)
<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>state</b>	TIMER channel complementary output enable state
<i>TIMER_CCXN_ENABLE</i>	channel complementary enable
<i>TIMER_CCXN_DISABLE</i>	channel complementary disable
<i>E</i>	

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output enable state */
timer_channel_complementary_output_state_config(TIMER0, TIMER_CH_0,
TIMER_CCXN_ENABLE);
```

### timer\_channel\_composite\_asymmetric\_pwm\_level\_config

The description of timer\_channel\_composite\_asymmetric\_pwm\_level\_config is shown as below:

**Table 3-738. Function timer\_channel\_composite\_asymmetric\_pwm\_level\_config**

Function name	timer_channel_composite_asymmetric_pwm_level_config
Function prototype	void timer_channel_composite_asymmetric_pwm_level_config(uint32_t timer_periph, uint16_t channel, ControlStatus newvalue);
Function descriptions	configure TIMER channel period point match moment OxCPRE level in the composite PWM or asymmetric PWM mode
Precondition	-
The called functions	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	TIMER peripheral selection
Input parameter{in}	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0,7,15,16)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0,7,15,16)
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx, x=0,7)
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx, x=0,7)
Input parameter{in}	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER0 channel 0 period point match moment OxCPRE level in the composite
PWM or asymmetric PWM mode */
```

```
timer_channel_composite_asymmetric_pwm_level_config(TIMER0, TIMER_CH_0,
ENABLE);
```

### timer\_channel\_output\_clear\_invalid\_selection

The description of timer\_channel\_output\_clear\_invalid\_selection is shown as below:

**Table 3-739. Function timer\_channel\_output\_clear\_invalid\_selection**

<b>Function name</b>	timer_channel_output_clear_invalid_selection
<b>Function prototype</b>	void timer_channel_output_clear_invalid_selection(uint32_t timer_periph, uint32_t event);
<b>Function descriptions</b>	output clear invalid event 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>event</b>	TIMER event
<i>TIMER_OCREFCLR_IN VALID_FLOW</i>	output clear invalid by next overflow or underflow event
<i>TIMER_OCREFCLR_IN VALID_UPDATE</i>	ouput clear invalid by next update event
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select update event as TIMER0 output clear invalid event*/
```

```
timer_channel_output_clear_invalid_selection(TIMER0,
TIMER_OCREFCLR_INVALID_UPDATE);
```

### timer\_channel\_input\_struct\_para\_init

The description of timer\_channel\_input\_struct\_para\_init is shown as below:

**Table 3-740. 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>	-

Input parameter{in}	
icpara	TIMER channel input parameter struct, the structure members can refer to <a href="#">Structure timer_ic_parameter_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER channel input parameter struct with a default value */
```

```
timer_ic_parameter_struct timer_icinitpara;
```

```
timer_channel_input_struct_para_init(&timer_icinitpara);
```

### timer\_input\_capture\_config

The description of timer\_input\_capture\_config is shown as below:

**Table 3-741. Function timer\_input\_capture\_config**

Function name	timer_input_capture_config
Function prototype	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
Function descriptions	configure TIMER input capture parameter
Precondition	-
The called functions	timer_channel_input_capture_prescaler_config
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0~4,7,15,16)	please refer to the following parameters
Input parameter{in}	
channel	TIMER channel
TIMER_CH_0	TIMER channel 0(TIMERx, x=0~4,7,15,16)
TIMER_CH_1	TIMER channel 1(TIMERx, x=0~4,7,15,16)
TIMER_CH_2	TIMER channel 2(TIMERx, x=0~4,7)
TIMER_CH_3	TIMER channel 3(TIMERx, x=0~4,7)
TIMER_MCH_0	TIMER multi mode channel 0(TIMERx, x=15,16)
Input parameter{in}	
icpara	TIMER channel input parameter struct, the structure members can refer to <a href="#">Structure timer_ic_parameter_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 input capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);

```

### timer\_channel\_input\_capture\_prescaler\_config

The description of timer\_channel\_input\_capture\_prescaler\_config is shown as below:

**Table 3-742. 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(x=0~4,7,15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0~4,7,15,16)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0~4,7,15,16)
<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)
<i>TIMER_MCH_0</i>	TIMER multi mode channel 0(TIMERx, x=15,16)
<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 3-743. 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(x=0~4,7,15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0~4,7,15,16)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0~4,7,15,16)
<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)
<i>TIMER_MCH_0</i>	TIMER multi mode channel 0(TIMERx, x=15,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	channel capture compare register value (0~0xFFFFFFFF)

Example:

```
/* read TIMER0 channel 0 capture compare register value */
```

```
uint32_t ch0_value = 0;
```

```
ch0_value = timer_channel_capture_value_register_read(TIMER0, TIMER_CH_0);
```

### timer\_input\_pwm\_capture\_config

The description of timer\_input\_pwm\_capture\_config is shown as below:

**Table 3-744. 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>TIMER<sub>x</sub>(x=0~4,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<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="#">Structure timer ic parameter struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure TIMER0 input pwm capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_pwm_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);

```

### timer\_hall\_mode\_config

The description of timer\_hall\_mode\_config is shown as below:

**Table 3-745. Function timer\_hall\_mode\_config**

<b>Function name</b>	timer_hall_mode_config
<b>Function prototype</b>	void timer_hall_mode_config(uint32_t timer_periph, uint32_t hallmode);
<b>Function descriptions</b>	configure TIMER hall sensor mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMER<sub>x</sub>(x=0~4,7,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>hallmode</b>	TIMER hall sensor mode state

<i>TIMER_HALLINTERFA CE_ENABLE</i>	TIMER hall sensor mode enable
<i>TIMER_HALLINTERFA CE_DISABLE</i>	TIMER hall sensor mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 hall sensor mode */
```

```
timer_hall_mode_config (TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

### timer\_multi\_mode\_channel\_output\_parameter\_struct\_init

The description of timer\_multi\_mode\_channel\_output\_parameter\_struct\_init is shown as below:

**Table 3-746. Function timer\_multi\_mode\_channel\_output\_parameter\_struct\_init**

<b>Function name</b>	timer_multi_mode_channel_output_parameter_struct_init
<b>Function prototype</b>	void timer_multi_mode_channel_output_parameter_struct_init(timer_omc_parameter_struct *omcpara);
<b>Function descriptions</b>	initialize TIMER multi mode channel output parameter struct
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>omcpara</b>	TIMER multi mode channel output parameter struct, the structure members can refer to <a href="#">Structure timer_omc_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize TIMER multi mode channel output parameter struct with a default value */
```

```
timer_omc_parameter_struct timer_omcinitpara;
```

```
timer_multi_mode_channel_output_parameter_struct_init(&timer_omcinitpara);
```

### timer\_multi\_mode\_channel\_output\_config

The description of timer\_multi\_mode\_channel\_output\_config is shown as below:

**Table 3-747. Function timer\_multi\_mode\_channel\_output\_config**

<b>Function name</b>	timer_multi_mode_channel_output_config
<b>Function prototype</b>	void timer_multi_mode_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_omc_parameter_struct *omcpara);
<b>Function descriptions</b>	configure TIMER multi mode channel output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_MCH_0</i>	TIMER multi mode channel 0(TIMERx, x=15,16)
<b>Input parameter{in}</b>	
<b>omcpara</b>	TIMER multi mode channel output parameter struct, the structure members can refer to <a href="#">Structure timer_omc_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER15 multi mode channel 0 output function */
```

```
timer_omc_parameter_struct timer_omcinitpara;
```

```
omcpara->outputmode = TIMER_MCH_MODE_INDEPENDENTLY;
```

```
omcpara->outputstate = TIMER_MCCX_ENABLE;
```

```
omcpara->ocpolarity = TIMER_OMC_POLARITY_HIGH;
```

```
timer_multi_mode_channel_output_parameter_struct_init(TIMER15,      TIMER_MCH_0,  
&timer_omcinitpara);
```

### timer\_multi\_mode\_channel\_mode\_config

The description of timer\_multi\_mode\_channel\_mode\_config is shown as below:

**Table 3-748. Function timer\_multi\_mode\_channel\_mode\_config**

<b>Function name</b>	timer_multi_mode_channel_mode_config
<b>Function prototype</b>	void timer_multi_mode_channel_mode_config(uint32_t timer_periph, uint32_t channel, uint32_t multi_mode_sel);
<b>Function descriptions</b>	multi mode channel mode select
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_MCH_0</i>	TIMER multi mode channel 0(TIMERx, x=15,16)
<b>Input parameter{in}</b>	
<b>multi_mode_sel</b>	multi mode channel mode selection
<i>TIMER_MCH_MODE_INDEPENDENTLY</i>	multi mode channel work in independently mode
<i>TIMER_MCH_MODE_COMPLEMENTARY</i>	multi mode channel work in complementary output mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER15 multi mode channel 0 mode */
```

```
timer_multi_mode_channel_mode_config(TIMER15, TIMER_MCH_0,
TIMER_MCH_MODE_INDEPENDENTLY);
```

### timer\_input\_trigger\_source\_select

The description of timer\_input\_trigger\_source\_select is shown as below:

**Table 3-749. Function timer\_input\_trigger\_source\_select**

<b>Function name</b>	timer_input_trigger_source_select
<b>Function prototype</b>	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);
<b>Function descriptions</b>	select TIMER input trigger source
<b>Precondition</b>	SMC[2:0] = 000
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~4,7,15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>intrigger</b>	input trigger source
<i>TIMER_SMCFG_TRGS_EL_ITI0</i>	internal trigger input 0 (ITI0, TIMERx(x=0..4,7,15,16))
<i>TIMER_SMCFG_TRGS_EL_ITI1</i>	internal trigger input 1 (ITI1, TIMERx(x=0..4,7,15,16))
<i>TIMER_SMCFG_TRGS_EL_ITI2</i>	internal trigger input 2 (ITI2, TIMERx(x=0..4,7,15,16))
<i>TIMER_SMCFG_TRGS</i>	internal trigger input 3 (ITI3, TIMERx(x=0..4,7,15,16))

<i>EL_ITI3</i>	
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	TI0 edge detector (CIOF_ED, TIMERx(x=0..4,7,15,16))
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	filtered channel channel 0 input (CIOFE0, TIMERx(x=0..4,7,15,16))
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI1FE1</i>	filtered channel channel 1 input (CI1FE1, TIMERx(x=0..4,7,15,16))
<i>TIMER_SMCFG_TRGS</i> <i>EL_ETIFP</i>	external trigger input filter output(ETIFP, TIMERx(x=0~4,7))
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI2FE2</i>	filtered channel 2 input(CI2FE2, TIMERx(x=0~4,7))
<i>TIMER_SMCFG_TRGS</i> <i>EL_CI3FE3</i>	filtered channel 3 input(CI3FE3, TIMERx(x=0~4,7))
<i>TIMER_SMCFG_TRGS</i> <i>EL_MCIOFEM0</i>	filtered multi mode channel 0 input(MCIOFEM0, TIMERx(x=15,16))
<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 3-750. Function timer\_master\_output0\_trigger\_source\_select**

<b>Function name</b>	timer_master_output0_trigger_source_select
<b>Function prototype</b>	void timer_master_output0_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
<b>Function descriptions</b>	select TIMER master mode output 0 trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~7,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>outrigger</b>	trigger output source
<i>TIMER_TRI_OUT0_SR</i> <i>C_RESET</i>	the UPG bit as trigger output 0 (TIMERx(x=0~7,15,16))
<i>TIMER_TRI_OUT0_SR</i> <i>C_ENABLE</i>	the counter enable signal as trigger output 0(TIMERx(x=0~7,15,16))

<i>TIMER_TRI_OUT0_SRC_UPDATE</i>	update event as trigger output 0(TIMERx(x=0~7,15,16))
<i>TIMER_TRI_OUT0_SRC_CH0</i>	a capture or a compare match occurred in channel 0 as trigger output 0 (TIMERx(x=0~4,7,15,16))
<i>TIMER_TRI_OUT0_SRC_O0CPRE</i>	O0CPRE as trigger output 0(TIMERx(x=0~4,7,15,16))
<i>TIMER_TRI_OUT0_SRC_O1CPRE</i>	O1CPRE as trigger output 0(TIMERx(x=0~4,7,15,16))
<i>TIMER_TRI_OUT0_SRC_O2CPRE</i>	O2CPRE as trigger output 0(TIMERx(x=0~4,7))
<i>TIMER_TRI_OUT0_SRC_O3CPRE</i>	O3CPRE as trigger output 0(TIMERx(x=0~4,7))
<i>TIMER_TRI_OUT0_SRC_DECODER_CLOCK</i>	decoder clock as trigger output 0(TIMERx(x=0,7,1,2,3,4))
<i>TIMER_TRI_OUT0_SRC_SYNC</i>	synchronization event as trigger output 0(TIMERx(x=0,7,15,16))
<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_OUT0_SRC_RESET);
```

### timer\_slave\_mode\_select

The description of timer\_slave\_mode\_select is shown as below:

**Table 3-751. Function timer\_slave\_mode\_select**

<b>Function name</b>	timer_slave_mode_select
<b>Function prototype</b>	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
<b>Function descriptions</b>	select TIMER slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~4,7,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>slavemode</b>	slave mode
<i>TIMER_SLAVE_MODE_DISABLE</i>	slave mode disable(TIMERx(x=0~4,7,15,16))
<i>TIMER_QUAD_DECODER_MODE0</i>	quadrature decoder mode 0(TIMERx(x=0~4,7))

<code>TIMER_QUAD_DECODER_MODE1</code>	quadrature decoder mode 1(TIMERx(x=0~4,7))
<code>TIMER_QUAD_DECODER_MODE2</code>	quadrature decoder mode 2(TIMERx(x=0~4,7))
<code>TIMER_SLAVE_MODE_RESTART</code>	restart mode(TIMERx(x=0~4,7,15,16))
<code>TIMER_SLAVE_MODE_PAUSE</code>	pause mode(TIMERx(x=0~4,7,15,16))
<code>TIMER_SLAVE_MODE_EVENT</code>	event mode(TIMERx(x=0~4,7,15,16))
<code>TIMER_SLAVE_MODE_EXTERNAL0</code>	external clock mode 0(TIMERx(x=0~4,7,15,16))
<code>TIMER_NONQUAD_DECODER_MODE0</code>	non-quadrature decoder mode 0(TIMERx(x=1~4))
<code>TIMER_NONQUAD_DECODER_MODE1</code>	non-quadrature decoder mode 1(TIMERx(x=1~4))
<code>TIMER_NONQUAD_DECODER_MODE2</code>	non-quadrature decoder mode 2(TIMERx(x=1~4))
<code>TIMER_NONQUAD_DECODER_MODE3</code>	non-quadrature decoder mode 3(TIMERx(x=1~4))
<code>TIMER_QUAD_DECODER_MODE3</code>	quadrature decoder mode 3(TIMERx(x=1~4))
<code>TIMER_QUAD_DECODER_MODE4</code>	quadrature decoder mode 4(TIMERx(x=1~4))
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER0 slave mode */
```

```
timer_slave_mode_select (TIMER0, TIMER_QUAD_DECODER_MODE0);
```

### timer\_master\_slave\_mode\_config

The description of timer\_master\_slave\_mode\_config is shown as below:

**Table 3-752. Function timer\_master\_slave\_mode\_config**

<b>Function name</b>	timer_master_slave_mode_config
<b>Function prototype</b>	void timer_master_slave_mode_config(uint32_t timer_periph, uint32_t masterslave);
<b>Function descriptions</b>	configure TIMER master slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~4,7,15,16)</i>	TIMER peripheral selection
Input parameter{in}	
<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
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 master slave mode */
timer_master_slave_mode_config(TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

### timer\_external\_trigger\_config

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

**Table 3-753. 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>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~4,7)</i>	TIMER peripheral selection
Input parameter{in}	
<b>extprescaler</b>	external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
Input parameter{in}	
<b>expolarity</b>	external trigger polarity



<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	external trigger filter control(0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 external trigger input */
```

```
timer_external_trigger_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 10);
```

### timer\_quadrature\_decoder\_mode\_config

The description of timer\_quadrature\_decoder\_mode\_config is shown as below:

**Table 3-754. Function timer\_quadrature\_decoder\_mode\_config**

<b>Function name</b>	timer_quadrature_decoder_mode_config
<b>Function prototype</b>	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
<b>Function descriptions</b>	configure TIMER quadrature decoder mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~4,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>decomode</b>	quadrature decoder mode
<i>TIMER_QUAD_DECODER_MODE0</i>	counter counts on CI0FE0 edge depending on CI1FE1 level (TIMERx(x=0..4,7))
<i>TIMER_QUAD_DECODER_MODE1</i>	counter counts on CI1FE1 edge depending on CI0FE0 level (TIMERx(x=0..4,7))
<i>TIMER_QUAD_DECODER_MODE2</i>	counter counts on both CI0FE0 and CI1FE1 edges depending on the level of the other input (TIMERx(x=0..4,7))
<i>TIMER_QUAD_DECODER_MODE3</i>	counter counts on CI0FE0 edge depending on CI0FE0 level (TIMERx(x=1..4))
<i>TIMER_QUAD_DECODER_MODE4</i>	counter counts on CI1FE1 edge depending on CI1FE1 level (TIMERx(x=1..4))
<b>Input parameter{in}</b>	
<b>ic0polarity</b>	CI0 input capture polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge

<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	active both edge
<b>Input parameter{in}</b>	
<b>ic1polarity</b>	CI1 input capture polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	active both edge
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config(TIMER0, TIMER_ENCODER_MODE0,  
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

### timer\_non\_quadrature\_decoder\_mode\_config

The description of timer\_non\_quadrature\_decoder\_mode\_config is shown as below:

**Table 3-755. Function timer\_non\_quadrature\_decoder\_mode\_config**

<b>Function name</b>	timer_non_quadrature_decoder_mode_config
<b>Function prototype</b>	void timer_non_quadrature_decoder_mode_config (uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
<b>Function descriptions</b>	configure TIMER non-quadrature decoder mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=1~4)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>decomode</b>	decoder mode
<i>TIMER_NONQUAD_DECODER_MODE0</i>	non-quadrature decoder mode 0
<i>TIMER_NONQUAD_DECODER_MODE1</i>	non-quadrature decoder mode 1
<i>TIMER_NONQUAD_DECODER_MODE2</i>	non-quadrature decoder mode 2

<i>TIMER_NONQUAD_DE CODER_MODE3</i>	non-quadrature decoder mode 3
<b>Input parameter{in}</b>	
<b>lc0polarity</b>	CI0 input capture polarity
<i>TIMER_IC_POLARITY_ RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_ FALLING</i>	capture falling edge
<i>TIMER_IC_POLARITY_ BOTH_EDGE</i>	active both edge
<b>Input parameter{in}</b>	
<b>lc1polarity</b>	CI1 input capture polarity
<i>TIMER_IC_POLARITY_ RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_ FALLING</i>	capture falling edge
<i>TIMER_IC_POLARITY_ BOTH_EDGE</i>	active both edge
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 non-quadrature decoder mode */
```

```
timer_decoder_mode_config(TIMER0, TIMER_NONQUAD_DECODER_MODE3,  
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

### timer\_internal\_clock\_config

The description of timer\_internal\_clock\_config is shown as below:

**Table 3-756. 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=0~4,7,15,16)</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 3-757. 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=0~4,7,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS EL_ITI0</i>	internal trigger input 0 (ITI0, TIMERx(x=0~4,7,15,16))
<i>TIMER_SMCFG_TRGS EL_ITI1</i>	internal trigger input 1 (ITI1, TIMERx(x=0~4,7,15,16))
<i>TIMER_SMCFG_TRGS EL_ITI2</i>	internal trigger input 2 (ITI2, TIMERx(x=0~4,7,15,16))
<i>TIMER_SMCFG_TRGS EL_ITI3</i>	internal trigger input 3 (ITI3, TIMERx(x=0~4,7,15,16))
<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 3-758. 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
TIMERx(x=0~4,7,15,16)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extrigger</b>	external trigger selection
TIMER_SMCFG_TRGS EL_CIOF_ED	TI0 edge detector (CIOF_ED, TIMERx(x=0~4,7,15,16))
TIMER_SMCFG_TRGS EL_CIOFE0	filtered channel channel 0 input (CIOFE0, TIMERx(x=0~4,7,15,16))
TIMER_SMCFG_TRGS EL_CI1FE1	filtered channel channel 1 input (CI1FE1, TIMERx(x=0~4,7,15,16))
TIMER_SMCFG_TRGS EL_CI2FE2	filtered channel 2 input(x=0~4,7,15,16))
TIMER_SMCFG_TRGS EL_CI3FE3	filtered channel 3 input(x=0~4,7,15,16))
TIMER_SMCFG_TRGS EL_MCIOFEM0	filtered multi mode channel 0 input(TIMERx(x=15,16))
<b>Input parameter{in}</b>	
<b>expolarity</b>	external trigger polarity
TIMER_IC_POLARITY_ RISING	active high or rising edge active
TIMER_IC_POLARITY_ FALLING	active low or falling edge active
TIMER_IC_POLARITY_ BOTH_EDGE	active both edge
<b>Input parameter{in}</b>	
<b>extfilter</b>	external trigger filter control(0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 the external trigger 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 3-759. 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
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>	
-	-
<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 3-760. Function timer\_external\_clock\_mode1\_config**

<b>Function name</b>	timer_external_clock_mode1_config
<b>Function prototype</b>	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external clock mode1
<b>Precondition</b>	-
<b>The called functions</b>	timer_external_trigger_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~4,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
<b>Input parameter{in}</b>	
<b>expolarity</b>	ETI external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	ETI external trigger filter control(0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

### timer\_external\_clock\_mode1\_disable

The description of timer\_external\_clock\_mode1\_disable is shown as below:

**Table 3-761. Function timer\_external\_clock\_mode1\_disable**

<b>Function name</b>	timer_external_clock_mode1_disable
<b>Function prototype</b>	void timer_external_clock_mode1_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER the external clock mode1

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~4,7)</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 3-762. 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 ccse1);
<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=0~4,7,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>ccse1</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 3-763. 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=0,7,15,16)</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\_commutation\_control\_shadow\_register\_config

The description of timer\_commutation\_control\_shadow\_register\_config is shown as below:

**Table 3-764. Function timer\_commutation\_control\_shadow\_register\_config**

<b>Function name</b>	timer_commutation_control_shadow_register_config
<b>Function prototype</b>	void timer_commutation_control_shadow_register_config(uint32_t timer_periph, uint16_t ccssel);
<b>Function descriptions</b>	configure commutation control shadow register update selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	

<b>cssel</b>	commutation control shadow register selection
<i>TIMER_CCUSEL_ENABLE</i>	the shadow registers update when the counter generates an overflow/ underflow event
<i>TIMER_CCUSEL_DISABLE</i>	the shadow registers update when the counter generates an overflow/ underflow event and the repetition counter value is zero
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure commutation control shadow register update selection */
```

```
timer_commutation_control_shadow_register_config (TIMER0, TIMER_CCUSEL_ENABLE);
```

### timer\_output\_match\_pulse\_select

The description of timer\_output\_match\_pulse\_select is shown as below:

**Table 3-765. Function timer\_output\_match\_pulse\_select**

<b>Function name</b>	timer_output_match_pulse_select
<b>Function prototype</b>	void timer_output_match_pulse_select(uint32_t timer_periph, uint32_t channel, uint16_t pulsesel);
<b>Function descriptions</b>	configure TIMER output match pulse selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0,7,15,16)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0,7,15,16)
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx, x=0,7)
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx, x=0,7)
<b>Input parameter{in}</b>	
<b>pulsesel</b>	output match pulse selection
<i>TIMER_PULSE_OUTPUT_T_NORMAL</i>	channel output normal
<i>TIMER_PULSE_OUTPUT_T_CNT_UP</i>	pulse output only when counting up
<i>TIMER_PULSE_OUTPUT_T_CNT_DOWN</i>	pulse output only when counting down
<i>TIMER_PULSE_OUTPUT_T_CNT_BOTH</i>	pulse output when counting up or down

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 match pulse selection */
```

```
timer_output_match_pulse_select          (TIMER0,          TIMER_CH_0,
TIMER_PULSE_OUTPUT_CNT_UP);
```

### timer\_channel\_composite\_pwm\_mode\_config

The description of timer\_channel\_composite\_pwm\_mode\_config is shown as below:

**Table 3-766. Function timer\_channel\_composite\_pwm\_mode\_config**

Function name	timer_channel_composite_pwm_mode_config
Function prototype	void timer_channel_composite_pwm_mode_config(uint32_t timer_periph, uint32_t channel, ControlStatus newvalue);
Function descriptions	configure the TIMER composite PWM mode
Precondition	-
The called functions	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	please refer to the following parameters
Input parameter{in}	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0,7,15,16)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0,7,15,16)
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx, x=0,7)
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx, x=0,7)
Input parameter{in}	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the TIMER composite PWM mode */
```

```
timer_channel_composite_pwm_mode_config (TIMER0, TIMER_CH_0, ENABLE);
```

### timer\_channel\_composite\_pwm\_mode\_output\_pulse\_value\_config

The description of timer\_channel\_composite\_pwm\_mode\_output\_pulse\_value\_config is shown as below:

**Table 3-767. Function timer\_channel\_composite\_pwm\_mode\_output\_pulse\_value\_config**

<b>Function name</b>	timer_channel_composite_pwm_mode_output_pulse_value_config
<b>Function prototype</b>	void timer_channel_composite_pwm_mode_output_pulse_value_config(uint32_t timer_periph, uint32_t channel, uint32_t pulse, uint32_t add_pulse);
<b>Function descriptions</b>	configure the TIMER composite PWM mode output pulse value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
TIMERx(x=0,7,15,16)	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
TIMER_CH_0	TIMER channel 0(TIMERx, x=0,7,15,16)
TIMER_CH_1	TIMER channel 1(TIMERx, x=0,7,15,16)
TIMER_CH_2	TIMER channel 2(TIMERx, x=0,7)
TIMER_CH_3	TIMER channel 3(TIMERx, x=0,7)
<b>Input parameter{in}</b>	
<b>pulse</b>	channel compare value(0~0xFFFFFFFF)
<b>Input parameter{in}</b>	
<b>add_pulse</b>	channel additional compare value(0~0xFFFFFFFF)
<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, 3999);
```

### timer\_channel\_asymmetric\_pwm\_pulse\_value\_config

The description of timer\_channel\_asymmetric\_pwm\_pulse\_config is shown as below:

**Table 3-768. Function timer\_channel\_asymmetric\_pwm\_pulse\_config**

<b>Function name</b>	timer_channel_asymmetric_pwm_pulse_config
<b>Function prototype</b>	void timer_channel_asymmetric_pwm_pulse_config (uint32_t timer_periph, uint32_t channel, uint32_t pulse, uint32_t add_pulse);

<b>Function descriptions</b>	configure the TIMER asymmetric PWM mode output pulse value
<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>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0,7)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0,7)
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx, x=0,7)
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx, x=0,7)
<b>Input parameter{in}</b>	
<b>pulse</b>	channel compare value(0~0xFFFF)
<b>Input parameter{in}</b>	
<b>add_pulse</b>	channel additional compare value(0~0xFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output pulse value */
```

```
timer_channel_asymmetric_pwm_pulse_config (TIMER0, TIMER_CH_0, 399, 3999);
```

### timer\_channel\_additional\_compare\_value\_config

The description of timer\_channel\_additional\_compare\_value\_config is shown as below:

**Table 3-769. Function timer\_channel\_additional\_compare\_value\_config**

<b>Function name</b>	timer_channel_additional_compare_value_config
<b>Function prototype</b>	void timer_channel_additional_compare_value_config(uint32_t timer_periph, uint16_t channel, uint32_t value);
<b>Function descriptions</b>	configure TIMER channel additional compare value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0,7,15,16)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0,7,15,16)
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx, x=0,7)

<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx, x=0,7)
<b>Input parameter{in}</b>	
<b>value</b>	channel additional compare value(0~0xFFFFFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 additional compare value */
```

```
timer_channel_additional_compare_value_config (TIMER0, TIMER_CH_0, 399);
```

### timer\_channel\_additional\_output\_update\_select

The description of timer\_channel\_additional\_output\_update\_select is shown as below:

**Table 3-770. Function timer\_channel\_additional\_output\_update\_select**

<b>Function name</b>	timer_channel_additional_output_update_select
<b>Function prototype</b>	void timer_channel_additional_output_update_select(uint32_t timer_periph, uint16_t channel, uint16_t update);
<b>Function descriptions</b>	select TIMER channel additional output register update source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0,7,15,16)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0,7,15,16)
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx, x=0,7)
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx, x=0,7)
<b>Input parameter{in}</b>	
<b>update</b>	channel additional output register update
<i>TIMER_ADD_UPDATE_BY_UPDATE</i>	channel additional compare value register update by update event
<i>TIMER_ADD_UPDATE_BY_COM_MATCH</i>	channel additional compare value register update by compare value match event
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER channel additional output register update source */
```

```
timer_channel_additional_output_update_select      (TIMER0,          TIMER_CH_0,
TIMER_ADD_UPDATE_BY_UPDATE);
```

### timer\_channel\_additional\_output\_shadow\_config

The description of timer\_channel\_additional\_output\_shadow\_config is shown as below:

**Table 3-771. Function timer\_channel\_additional\_output\_shadow\_config**

Function name	timer_channel_additional_output_shadow_config
Function prototype	void timer_channel_additional_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t aocshadow);
Function descriptions	configure TIMER channel additional output shadow function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0,7,15,16)	please refer to the following parameters
Input parameter{in}	
channel	TIMER channel
TIMER_CH_0	TIMER channel 0(TIMERx, x=0,7,15,16)
TIMER_CH_1	TIMER channel 1(TIMERx, x=0,7,15,16)
TIMER_CH_2	TIMER channel 2(TIMERx, x=0,7)
TIMER_CH_3	TIMER channel 3(TIMERx, x=0,7)
Input parameter{in}	
aocshadow	channel additional output compare shadow
TIMER_ADD_SHADOW_ENABLE	channel additional output compare shadow enable
TIMER_ADD_SHADOW_DISABLE	channel additional output compare shadow disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*configure TIMER0 channel 0 additional output shadow function */
```

```
timer_channel_additional_output_shadow_config      (TIMER0,          TIMER_CH_0,
TIMER_OC_SHADOW_ENABLE);
```

### timer\_channel\_additional\_compare\_value\_read

The description of timer\_channel\_additional\_compare\_value\_read is shown as below:

**Table 3-772. Function timer\_channel\_additional\_compare\_value\_read**

<b>Function name</b>	timer_channel_additional_compare_value_read
<b>Function prototype</b>	uint32_t timer_channel_additional_compare_value_read(uint32_t timer_periph, uint16_t channel);
<b>Function descriptions</b>	read TIMER channel additional compare value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx, x=0,7,15,16)
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx, x=0,7,15,16)
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx, x=0,7)
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx, x=0,7)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	channel additional compare value, 0~0xFFFFFFFF

Example:

```
/* get TIMER autoreload register value */
```

```
uint32_t i = 0;
```

```
i = timer_channel_additional_compare_value_read(TIMER0, TIMER_CH_0);
```

### timer\_break\_external\_source\_config

The description of timer\_break\_external\_source\_config is shown as below:

**Table 3-773. Function timer\_break\_external\_source\_config**

<b>Function name</b>	timer_break_external_source_config
<b>Function prototype</b>	void timer_break_external_source_config(uint32_t timer_periph, uint32_t break_src, ControlStatus newvalue);
<b>Function descriptions</b>	configure the TIMER break source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>break_src</b>	break source
<i>TIMER_BRKIN0</i>	BRKIN0 alternate function input enable, TIMERx(x=0,7,15,16)



<i>TIMER_BRKCOMP0</i>	CMP0 input enable, TIMERx(x=0,7,15,16)
<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:

```
/* configure the TIMER break source */
```

```
timer_break_external_source_config(TIMER0, TIMER_BRKIN0, ENABLE);
```

### timer\_break\_external\_polarity\_config

The description of timer\_break\_external\_polarity\_config is shown as below:

**Table 3-774. Function timer\_break\_external\_polarity\_config**

<b>Function name</b>	timer_break_external_polarity_config
<b>Function prototype</b>	void timer_break_external_polarity_config(uint32_t timer_periph, uint32_t break_src, uint16_t bkinpolarity);
<b>Function descriptions</b>	configure TIMER break polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>break_src</b>	break source
<i>TIMER_BRKIN0</i>	BRKIN0 alternate function input enable, TIMERx(x=0,7,15,16)
<i>TIMER_BRKCOMP0</i>	CMP0 input enable, TIMERx(x=0,7,15,16)
<b>Input parameter{in}</b>	
<b>bkinpolarity</b>	break polarity
<i>TIMER_BRKIN_POLARITY_LOW</i>	active low
<i>TIMER_BRKIN_POLARITY_HIGH</i>	active high
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the TIMER break polarity */
```

```
timer_break_external_polarity_config(TIMER0,                                TIMER_BRKIN0,
TIMER_BRKIN_POLARITY_HIGH);
```

### timer\_dead\_time\_falling\_edge\_config

The description of timer\_dead\_time\_falling\_edge\_config is shown as below:

**Table 3-775. Function timer\_dead\_time\_falling\_edge\_config**

Function name	timer_dead_time_falling_edge_config
Function prototype	void timer_dead_time_falling_edge_config(uint32_t timer_periph, uint32_t deadtime);
Function descriptions	
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0,7)	TIMER peripheral selection
Input parameter{in}	
deadtime	dead time (0~255)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the TIMER falling edge dead time */
```

```
timer_dead_time_falling_edge_config (TIMER0, 255);
```

### timer\_dead\_time\_different\_config

The description of timer\_dead\_time\_different\_config is shown as below:

**Table 3-776. Function timer\_dead\_time\_different\_config**

Function name	timer_dead_time_different_config
Function prototype	void timer_dead_time_different_config(uint32_t timer_periph, ControlStatus newvalue);
Function descriptions	configure the TIMER dead time different function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0,7)	please refer to the following parameters
Input parameter{in}	

<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:

```
/* configure the TIMER dead time different function */
```

```
timer_dead_time_different_config (TIMER0, ENABLE);
```

### timer\_channel\_break\_control\_config

The description of timer\_channel\_break\_control\_config is shown as below:

**Table 3-777. Function timer\_channel\_break\_control\_config**

<b>Function name</b>	timer_channel_break_control_config
<b>Function prototype</b>	void timer_channel_break_control_config(uint32_t timer_periph, uint32_t channel, ControlStatus newvalue);
<b>Function descriptions</b>	configure the TIMER channel break control 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>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<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
<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:

```
/* configure TIMER0 channel 0 break function */
```

timer\_channel\_break\_control\_config (TIMER0, TIMER\_CH\_0, ENABLE);

### timer\_channel\_dead\_time\_config

The description of timer\_channel\_dead\_time\_config is shown as below:

**Table 3-778. Function timer\_channel\_dead\_time\_config**

<b>Function name</b>	timer_channel_dead_time_config
<b>Function prototype</b>	void timer_channel_dead_time_config(uint32_t timer_periph, uint32_t channel, ControlStatus newvalue);
<b>Function descriptions</b>	configure the TIMER channel free dead time 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>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<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
<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:

```
/* configure TIMER0 channel 0 free dead time function */
```

```
timer_channel_dead_time_config (TIMER0, TIMER_CH_0, ENABLE);
```

### timer\_channel\_break\_config

The description of timer\_channel\_break\_config is shown as below:

**Table 3-779. Function timer\_channel\_break\_config**

<b>Function name</b>	timer_channel_break_config
<b>Function prototype</b>	void timer_channel_break_config(uint32_t timer_periph, uint32_t broken, uint32_t breakpolarity, uint32_t break0filter);
<b>Function descriptions</b>	configure the TIMER channel break
<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>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>broken</b>	TIMER channel break enable
<i>TIMER_CH_BREAK_ENABLE</i>	channel BREAK input signal enable
<i>TIMER_CH_BREAK_DISABLE</i>	channel BREAK input signal disable
<b>Input parameter{in}</b>	
<b>breakpolarity</b>	channel BREAK input signal polarity
<i>TIMER_CH_BREAK_POLARITY_HIGH</i>	channel BREAK input active high
<i>TIMER_CH_BREAK_POLARITY_LOW</i>	channel BREAK input active low
<b>Input parameter{in}</b>	
<b>break0filter</b>	channel BREAK input filter(0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel break */
```

```
timer_channel_break_config (TIMER0, TIMER_CH_0, TIMER_CH_BREAK_ENABLE ,  
TIMER_CH_BREAK_POLARITY_LOW, 15);
```

### timer\_channel\_break\_external\_status\_config

The description of timer\_channel\_break\_external\_status\_config is shown as below:

**Table 3-780. Function timer\_channel\_break\_external\_status\_config**

<b>Function name</b>	timer_channel_break_external_status_config
<b>Function prototype</b>	void timer_channel_break_external_status_config(uint32_t timer_periph, uint32_t channel, ControlStatus newvalue);
<b>Function descriptions</b>	configure TIMER channel break external input 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>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel

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

Example:

```
/* configure TIMER0 channel0 break external input enable */
```

```
timer_channel_break_external_status_config(TIMER0, TIMER_CH_0, ENABLE);
```

### timer\_channel\_break\_external\_polarity\_config

The description of timer\_channel\_break\_external\_polarity\_config is shown as below:

**Table 3-781. Function timer\_channel\_break\_external\_polarity\_config**

<b>Function name</b>	timer_channel_break_external_polarity_config
<b>Function prototype</b>	void timer_channel_break_external_polarity_config(uint32_t timer_periph, uint32_t channel, uint16_t bkinpolarity);
<b>Function descriptions</b>	configure TIMER channel break external input 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>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<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>bkinpolarity</b>	channel x BREAK input signal polarity
<i>TIMER_CHx_BREAK_IN_INV</i>	channel x BREAK input signal will be inverted
<i>TIMER_CHx_BREAK_IN_NOT_INV</i>	channel x BREAK input signal will not be inverted
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* configure TIMER0 channel0 break external input polarity */
```

```
timer_channel_break_external_polarity_config (TIMER0,          TIMER_CH_0,
TIMER_CHx_BREAK_IN_NOT_INV);
```

### timer\_channel\_primary\_output\_config

The description of timer\_channel\_primary\_output\_config is shown as below:

**Table 3-782. Function timer\_channel\_primary\_output\_config**

<b>Function name</b>	timer_channel_primary_output_config
<b>Function prototype</b>	void timer_channel_primary_output_config(uint32_t timer_periph, uint32_t channel, ControlStatus newvalue);
<b>Function descriptions</b>	configure TIMER channel 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>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	TIMER channel
<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>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel0 primary output function */
```

```
timer_channel_primary_output_config (TIMER0, TIMER_CH_0, ENABLE);
```

### timer\_channel\_break\_period\_config

The description of timer\_channel\_break\_period\_config is shown as below:

Table 3-783. Function timer\_channel\_break\_period\_config

Function name	timer_channel_break_period_config
Function prototype	void timer_channel_break_period_config(uint32_t timer_periph, uint16_t value);
Function descriptions	configure TIMER channel break period value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0,7)	TIMER peripheral selection
Input parameter{in}	
value	channel break period value, 0~0xFFFF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel break period value */
timer_channel_break_period_config(TIMER0, 0xFFFF);
```

### timer\_watchdog\_value\_config

The description of timer\_watchdog\_value\_config is shown as below:

Table 3-784. Function timer\_watchdog\_value\_config

Function name	timer_watchdog_value_config
Function prototype	void timer_watchdog_value_config(uint32_t timer_periph, uint32_t value);
Function descriptions	configure TIMER autoreload register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=1~4)	TIMER peripheral selection
Input parameter{in}	
value	watchdog counter period value, 0~0xFFFFFFFF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure quadrature decoder signal disconnection detection watchdog value */
```



```
timer_watchdog_value_config(TIMER0, 3000);
```

## timer\_watchdog\_value\_read

The description of timer\_watchdog\_value\_read is shown as below:

**Table 3-785. Function timer\_watchdog\_value\_read**

<b>Function name</b>	timer_watchdog_value_read
<b>Function prototype</b>	uint32_t timer_watchdog_value_read(uint32_t timer_periph);
<b>Function descriptions</b>	read quadrature decoder signal disconnection detection watchdog value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=1~4)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	watchdog counter period register value, 0~0xFFFFFFFF

Example:

```
/* read quadrature decoder signal disconnection detection watchdog value */
```

```
uint32_t i = 0;
```

```
i = timer_watchdog_value_read(TIMER0);
```

## timer\_decoder\_disconnection\_detection\_config

The description of timer\_decoder\_disconnection\_detection\_config is shown as below:

**Table 3-786. Function timer\_decoder\_disconnection\_detection\_config**

<b>Function name</b>	timer_decoder_disconnection_detection_config
<b>Function prototype</b>	void timer_decoder_disconnection_detection_config(uint32_t timer_periph, ControlStatus newvalue);
<b>Function descriptions</b>	configure quadrature decoder signal disconnection detection function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=1~4)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* configure quadrature decoder signal disconnection detection function */
```

```
timer_decoder_disconnection_detection_config(TIMER0, ENABLE);
```

### timer\_decoder\_jump\_detection\_config

The description of timer\_decoder\_jump\_detection\_config is shown as below:

**Table 3-787. Function timer\_decoder\_jump\_detection\_config**

<b>Function name</b>	timer_decoder_jump_detection_config
<b>Function prototype</b>	void timer_decoder_jump_detection_config(uint32_t timer_periph, ControlStatus newvalue);
<b>Function descriptions</b>	configure quadrature decoder signal jump detection function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=1~4)</i>	please refer to the following parameters
<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:

```
/* configure quadrature decoder signal jump detection function */
```

```
timer_decoder_jump_detection_config(TIMER0, ENABLE);
```

### timer\_counter\_initial\_register\_config

The description of timer\_counter\_initial\_register\_config is shown as below:

**Table 3-788. Function timer\_counter\_initial\_register\_config**

<b>Function name</b>	timer_counter_initial_register_config
<b>Function prototype</b>	void timer_counter_initial_register_config(uint32_t timer_periph, ControlStatus newvalue);
<b>Function descriptions</b>	configure counter initial value register

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	please refer to the following parameters
<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:

```
/* configure counter initial value register */
```

```
timer_counter_initial_register_config (TIMER0, ENABLE);
```

### timer\_counter\_initial\_config

The description of timer\_counter\_initial\_config is shown as below:

**Table 3-789. Function timer\_counter\_initial\_config**

<b>Function name</b>	timer_counter_initial_config
<b>Function prototype</b>	void timer_counter_initial_config(uint32_t timer_periph, uint32_t value, uint32_t direction);
<b>Function descriptions</b>	configure counter initial value and direction
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>value</b>	counter initial value, 0~0xFFFF
<b>Input parameter{in}</b>	
<b>direction</b>	counter initial direction
<i>TIMER_INITIAL_DIRECTION_DOWN</i>	counter count down when synchronizat on event occurs
<i>TIMER_INITIAL_DIRECTION_UP</i>	counter count up when synchronizat on event occurs
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure counter initial value and direction */
```

```
timer_counter_initial_config (TIMER0, TIMER_INITIAL_DIRECTION_UP);
```

### timer\_synchronization\_event\_generate

The description of timer\_synchronization\_event\_generate is shown as below:

**Table 3-790. Function timer\_synchronization\_event\_generate**

<b>Function name</b>	timer_synchronization_event_generate
<b>Function prototype</b>	void timer_synchronization_event_generate(uint32_t timer_periph);
<b>Function descriptions</b>	generate soft synchronization event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7,15,16)</i>	please refer to the following parameters
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* generate soft synchronization event */
```

```
timer_synchronization_event_generate (TIMER0);
```

### timer\_flag\_get

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

**Table 3-791. 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(x=0~7,15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the TIMER flags
<i>TIMER_FLAG_UP</i>	update flag, TIMERx(x=0~7,15,16)
<i>TIMER_FLAG_CH0</i>	channel 0 capture or compare flag, TIMERx(x=0~4,7,15,16)
<i>TIMER_FLAG_CH1</i>	channel 1 capture or compare flag, TIMERx(x=0~4,7,15,16)

<i>TIMER_FLAG_CH2</i>	channel 2 capture or compare flag, $TIMERx(x=0\sim4,7)$
<i>TIMER_FLAG_CH3</i>	channel 3 capture or compare flag, $TIMERx(x=0\sim4,7)$
<i>TIMER_FLAG_CMT</i>	channel commutation flag, $TIMERx(x=0,7,15,16)$
<i>TIMER_FLAG_TRG</i>	trigger flag, $TIMERx(x=0\sim4,7,15,16)$
<i>TIMER_FLAG_BRK0</i>	BREAK0 flag, $TIMERx(x=0,7,15,16)$
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, $TIMERx(x=0\sim4,7,15,16)$
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, $TIMERx(x=0\sim4,7,15,16)$
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, $TIMERx(x=0\sim4,7)$
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, $TIMERx(x=0\sim4,7)$
<i>TIMER_FLAG_SYSB</i>	system source break flag, $TIMERx(x=0,7)$
<i>TIMER_FLAG_DECJ</i>	quadrature decoder signal jump flag, $TIMERx(x=1\sim4)$
<i>TIMER_FLAG_DECDIS</i>	quadrature decoder signal disconnection flag, $TIMERx(x=1\sim4)$
<i>TIMER_FLAG_MCH0</i>	multi mode channel 0 capture or compare flag, $TIMERx(x=15,16)$
<i>TIMER_FLAG_MCH0O</i>	multi mode channel 0 overcapture flag, $TIMERx(x=15,16)$
<i>TIMER_FLAG_CH0CO</i> <i>MADD</i>	channel 0 additional compare flag, $TIMERx(x=0,7,15,16)$
<i>TIMER_FLAG_CH1CO</i> <i>MADD</i>	channel 1 additional compare flag, $TIMERx(x=0,7,15,16)$
<i>TIMER_FLAG_CH2CO</i> <i>MADD</i>	channel 2 additional compare flag, $TIMERx(x=0,7)$
<i>TIMER_FLAG_CH3CO</i> <i>MADD</i>	channel 3 additional compare flag, $TIMERx(x=0,7)$
<i>TIMER_FLAG_CH0BRK</i> <i>M</i>	channel 0 BREAK multi-period interrupt flag, $TIMERx(x=0,7)$
<i>TIMER_FLAG_CH1BRK</i> <i>M</i>	channel 1 BREAK multi-period interrupt flag, $TIMERx(x=0,7)$
<i>TIMER_FLAG_CH2BRK</i> <i>M</i>	channel 2 BREAK multi-period interrupt flag, $TIMERx(x=0,7)$
<i>TIMER_FLAG_CH0BRK</i>	channel 0 BREAK interrupt flag, $TIMERx(x=0,7)$
<i>TIMER_FLAG_CH1BRK</i>	channel 1 BREAK interrupt flag, $TIMERx(x=0,7)$
<i>TIMER_FLAG_CH2BRK</i>	channel 2 BREAK interrupt flag, $TIMERx(x=0,7)$
<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 3-792. Function timer\_flag\_clear**

<b>Function name</b>	timer_flag_clear
<b>Function prototype</b>	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
<b>Function descriptions</b>	clear TIMER flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~7,15,16)</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(x=0~7,15,16)</i>
<i>TIMER_FLAG_CH0</i>	channel 0 capture or compare flag, <i>TIMERx(x=0~4,7,15,16)</i>
<i>TIMER_FLAG_CH1</i>	channel 1 capture or compare flag, <i>TIMERx(x=0~4,7,15,16)</i>
<i>TIMER_FLAG_CH2</i>	channel 2 capture or compare flag, <i>TIMERx(x=0~4,7)</i>
<i>TIMER_FLAG_CH3</i>	channel 3 capture or compare flag, <i>TIMERx(x=0~4,7)</i>
<i>TIMER_FLAG_CMT</i>	channel commutation flag, <i>TIMERx(x=0,7,15,16)</i>
<i>TIMER_FLAG_TRG</i>	trigger flag, <i>TIMERx(x=0~4,7,15,16)</i>
<i>TIMER_FLAG_BRK0</i>	BREAK0 flag, <i>TIMERx(x=0,7,15,16)</i>
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, <i>TIMERx(x=0~4,7,15,16)</i>
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, <i>TIMERx(x=0~4,7,15,16)</i>
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, <i>TIMERx(x=0~4,7)</i>
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, <i>TIMERx(x=0~4,7)</i>
<i>TIMER_FLAG_SYSB</i>	system source break flag, <i>TIMERx(x=0,7)</i>
<i>TIMER_FLAG_DECJ</i>	quadrature decoder signal jump flag, <i>TIMERx(x=1~4)</i>
<i>TIMER_FLAG_DECDIS</i>	quadrature decoder signal disconnection flag, <i>TIMERx(x=1~4)</i>
<i>TIMER_FLAG_MCH0</i>	multi mode channel 0 capture or compare flag, <i>TIMERx(x=15,16)</i>
<i>TIMER_FLAG_MCH0O</i>	multi mode channel 0 overcapture flag, <i>TIMERx(x=15,16)</i>
<i>TIMER_FLAG_CH0CO</i> <i>MADD</i>	channel 0 additional compare flag, <i>TIMERx(x=0,7,15,16)</i>
<i>TIMER_FLAG_CH1CO</i> <i>MADD</i>	channel 1 additional compare flag, <i>TIMERx(x=0,7,15,16)</i>
<i>TIMER_FLAG_CH2CO</i> <i>MADD</i>	channel 2 additional compare flag, <i>TIMERx(x=0,7)</i>
<i>TIMER_FLAG_CH3CO</i> <i>MADD</i>	channel 3 additional compare flag, <i>TIMERx(x=0,7)</i>
<i>TIMER_FLAG_CH0BRK</i> <i>M</i>	channel 0 BREAK multi-period interrupt flag, <i>TIMERx(x=0,7)</i>
<i>TIMER_FLAG_CH1BRK</i> <i>M</i>	channel 1 BREAK multi-period interrupt flag, <i>TIMERx(x=0,7)</i>

<i>TIMER_FLAG_CH2BRK</i> <i>M</i>	channel 2 BREAK multi-period interrupt flag, $TIMERx(x=0,7)$
<i>TIMER_FLAG_CH0BRK</i>	channel 0 BREAK interrupt flag, $TIMERx(x=0,7)$
<i>TIMER_FLAG_CH1BRK</i>	channel 1 BREAK interrupt flag, $TIMERx(x=0,7)$
<i>TIMER_FLAG_CH2BRK</i>	channel 2 BREAK interrupt flag, $TIMERx(x=0,7)$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER0 update flags */
```

```
timer_flag_clear(TIMER0, TIMER_FLAG_UP);
```

### timer\_interrupt\_enable

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

**Table 3-793. 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(x=0~7, 15, 16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	timer interrupt enable source
<i>TIMER_INT_UP</i>	update interrupt, $TIMERx(x=0~7, 15, 16)$
<i>TIMER_INT_CH0</i>	channel 0 capture or compare interrupt, $TIMERx(x=0~4, 7, 15, 16)$
<i>TIMER_INT_CH1</i>	channel 1 capture or compare interrupt, $TIMERx(x=0~4, 7, 15, 16)$
<i>TIMER_INT_CH2</i>	channel 2 capture or compare interrupt, $TIMERx(x=0~4, 7)$
<i>TIMER_INT_CH3</i>	channel 3 capture or compare interrupt, $TIMERx(x=0~4, 7)$
<i>TIMER_INT_CMT</i>	channel commutation interrupt, $TIMERx(x=0, 7, 15, 16)$
<i>TIMER_INT_TRG</i>	trigger interrupt, $TIMERx(x=0~4, 7, 15, 16)$
<i>TIMER_INT_BRK</i>	break interrupt, $TIMERx(x=0, 7, 15, 16)$
<i>TIMER_INT_DECJ</i>	quadrature decoder signal jump interrupt, $TIMERx(x=1~4)$
<i>TIMER_INT_DECDIS</i>	quadrature decoder signal disconnection interrupt, $TIMERx(x=1~4)$
<i>TIMER_INT_MCH0</i>	multi mode channel 0 capture or compare interrupt, $TIMERx(x=15, 16)$
<i>TIMER_INT_CH0COMA</i> <i>DD</i>	channel 0 additional compare interrupt, $TIMERx(x=0, 7, 15, 16)$
<i>TIMER_INT_CH0COMA</i> <i>DD</i>	channel 1 additional compare interrupt, $TIMERx(x=0, 7, 15, 16)$

<i>TIMER_INT_CH0COMA</i> <i>DD</i>	channel 2 additional compare interrupt, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_CH0COMA</i> <i>DD</i>	channel 3 additional compare interrupt, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_CH0BRKM</i>	channel 0 BREAK multi-period interrupt, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_CH1BRKM</i>	channel 1 BREAK multi-period interrupt, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_CH2BRKM</i>	channel 2 BREAK multi-period interrupt, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_CH0BRK</i>	channel 0 BREAK interrupt, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_CH1BRK</i>	channel 1 BREAK interrupt, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_CH2BRK</i>	channel 2 BREAK interrupt, <i>TIMERx</i> ( <i>x</i> =0,7)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER0 update interrupt */
```

```
timer_interrupt_enable (TIMER0, TIMER_INT_UP);
```

### timer\_interrupt\_disable

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

**Table 3-794. Function timer\_interrupt\_disable**

<b>Function name</b>	timer_interrupt_disable
<b>Function prototype</b>	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable the TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> ( <i>x</i> =0~7,15,16)	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	timer interrupt disable source
<i>TIMER_INT_UP</i>	update interrupt, <i>TIMERx</i> ( <i>x</i> =0~7,15,16)
<i>TIMER_INT_CH0</i>	channel 0 capture or compare interrupt, <i>TIMERx</i> ( <i>x</i> =0~4,7,15,16)
<i>TIMER_INT_CH1</i>	channel 1 capture or compare interrupt, <i>TIMERx</i> ( <i>x</i> =0~4,7,15,16)
<i>TIMER_INT_CH2</i>	channel 2 capture or compare interrupt, <i>TIMERx</i> ( <i>x</i> =0~4,7)
<i>TIMER_INT_CH3</i>	channel 3 capture or compare interrupt, <i>TIMERx</i> ( <i>x</i> =0~4,7)
<i>TIMER_INT_CMT</i>	channel commutation interrupt, <i>TIMERx</i> ( <i>x</i> =0,7,15,16)
<i>TIMER_INT_TRG</i>	trigger interrupt, <i>TIMERx</i> ( <i>x</i> =0~4,7,15,16)
<i>TIMER_INT_BRK</i>	break interrupt, <i>TIMERx</i> ( <i>x</i> =0,7,15,16)
<i>TIMER_INT_DECJ</i>	quadrature decoder signal jump interrupt, <i>TIMERx</i> ( <i>x</i> =1~4)
<i>TIMER_INT_DECDIS</i>	quadrature decoder signal disconnection interrupt, <i>TIMERx</i> ( <i>x</i> =1~4)



<i>TIMER_INT_MCH0</i>	multi mode channel 0 capture or compare interrupt, $TIMERx(x=15,16)$
<i>TIMER_INT_CH0COMA DD</i>	channel 0 additional compare interrupt, $TIMERx(x=0,7,15,16)$
<i>TIMER_INT_CH0COMA DD</i>	channel 1 additional compare interrupt, $TIMERx(x=0,7,15,16)$
<i>TIMER_INT_CH0COMA DD</i>	channel 2 additional compare interrupt, $TIMERx(x=0,7)$
<i>TIMER_INT_CH0COMA DD</i>	channel 3 additional compare interrupt, $TIMERx(x=0,7)$
<i>TIMER_INT_CH0BRKM</i>	channel 0 BREAK multi-period interrupt, $TIMERx(x=0,7)$
<i>TIMER_INT_CH1BRKM</i>	channel 1 BREAK multi-period interrupt, $TIMERx(x=0,7)$
<i>TIMER_INT_CH2BRKM</i>	channel 2 BREAK multi-period interrupt, $TIMERx(x=0,7)$
<i>TIMER_INT_CH0BRK</i>	channel 0 BREAK interrupt, $TIMERx(x=0,7)$
<i>TIMER_INT_CH1BRK</i>	channel 1 BREAK interrupt, $TIMERx(x=0,7)$
<i>TIMER_INT_CH2BRK</i>	channel 2 BREAK interrupt, $TIMERx(x=0,7)$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER0 update interrupt */
```

```
timer_interrupt_disable(TIMER0, TIMER_INT_UP);
```

### timer\_interrupt\_flag\_get

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

**Table 3-795. Function timer\_interrupt\_flag\_get**

<b>Function name</b>	timer_interrupt_flag_get
<b>Function prototype</b>	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t int_flag);
<b>Function descriptions</b>	get timer interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~7,15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>int_flag</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, $TIMERx(x=0~7,15,16)$
<i>TIMER_INT_FLAG_CH0</i>	channel 0 capture or compare interrupt flag, $TIMERx(x=0~4,7,15,16)$
<i>TIMER_INT_FLAG_CH1</i>	channel 1 capture or compare interrupt flag, $TIMERx(x=0~4,7,15,16)$
<i>TIMER_INT_FLAG_CH2</i>	channel 2 capture or compare interrupt flag, $TIMERx(x=0~4,7)$

<i>TIMER_INT_FLAG_CH3</i>	channel 3 capture or compare 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> =0,7,15,16)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> ( <i>x</i> =0~4,7,15,16)
<i>TIMER_INT_FLAG_BRK0</i>	BREAK0 interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7,15,16)
<i>TIMER_INT_FLAG_SYSB</i>	system source break interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_FLAG_DECJ</i>	quadrature decoder signal jump interrupt flag, <i>TIMERx</i> ( <i>x</i> =1~4)
<i>TIMER_INT_FLAG_DECDIS</i>	quadrature decoder signal disconnection interrupt flag, <i>TIMERx</i> ( <i>x</i> =1~4)
<i>TIMER_INT_FLAG_MCH0</i>	multi mode channel 0 capture or compare interrupt flag, <i>TIMERx</i> ( <i>x</i> =15,16)
<i>TIMER_INT_FLAG_CH0COMADD</i>	channel 0 additional compare interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7,15,16)
<i>TIMER_INT_FLAG_CH1COMADD</i>	channel 1 additional compare interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7,15,16)
<i>TIMER_INT_FLAG_CH2COMADD</i>	channel 2 additional compare interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_FLAG_CH3COMADD</i>	channel 3 additional compare interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_FLAG_CH0BRKM</i>	channel 0 BREAK multi-period interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_FLAG_CH1BRKM</i>	channel 1 BREAK multi-period interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_FLAG_CH2BRKM</i>	channel 2 BREAK multi-period interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_FLAG_CH0BRK</i>	channel 0 BREAK interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_FLAG_CH1BRK</i>	channel 1 BREAK interrupt flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_INT_FLAG_CH2BRK</i>	channel 2 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 3-796. Function timer\_interrupt\_flag\_clear**

<b>Function name</b>	timer_interrupt_flag_clear
<b>Function prototype</b>	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t int_flag);
<b>Function descriptions</b>	clear TIMER interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0~7,15,16)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>int_flag</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, <i>TIMERx(x=0~7,15,16)</i>
<i>TIMER_INT_FLAG_CH0</i>	channel 0 capture or compare interrupt flag, <i>TIMERx(x=0~4,7,15,16)</i>
<i>TIMER_INT_FLAG_CH1</i>	channel 1 capture or compare interrupt flag, <i>TIMERx(x=0~4,7,15,16)</i>
<i>TIMER_INT_FLAG_CH2</i>	channel 2 capture or compare interrupt flag, <i>TIMERx(x=0~4,7)</i>
<i>TIMER_INT_FLAG_CH3</i>	channel 3 capture or compare interrupt flag, <i>TIMERx(x=0~4,7)</i>
<i>TIMER_INT_FLAG_CMT</i>	channel commutation interrupt flag, <i>TIMERx(x=0,7,15,16)</i>
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx(x=0~4,7,15,16)</i>
<i>TIMER_INT_FLAG_BRK0</i>	BREAK0 interrupt flag, <i>TIMERx(x=0,7,15,16)</i>
<i>TIMER_INT_FLAG_SYSB</i>	system source break interrupt flag, <i>TIMERx(x=0,7)</i>
<i>TIMER_INT_FLAG_DECJ</i>	quadrature decoder signal jump interrupt flag, <i>TIMERx(x=1~4)</i>
<i>TIMER_INT_FLAG_DECDIS</i>	quadrature decoder signal disconnection interrupt flag, <i>TIMERx(x=1~4)</i>
<i>TIMER_INT_FLAG_MCH0</i>	multi mode channel 0 capture or compare interrupt flag, <i>TIMERx(x=15,16)</i>
<i>TIMER_INT_FLAG_CH0COMADD</i>	channel 0 additional compare interrupt flag, <i>TIMERx(x=0,7,15,16)</i>
<i>TIMER_INT_FLAG_CH1COMADD</i>	channel 1 additional compare interrupt flag, <i>TIMERx(x=0,7,15,16)</i>
<i>TIMER_INT_FLAG_CH2COMADD</i>	channel 2 additional compare interrupt flag, <i>TIMERx(x=0,7)</i>
<i>TIMER_INT_FLAG_CH3COMADD</i>	channel 3 additional compare interrupt flag, <i>TIMERx(x=0,7)</i>
<i>TIMER_INT_FLAG_CH0BRKM</i>	channel 0 BREAK multi-period interrupt flag, <i>TIMERx(x=0,7)</i>
<i>TIMER_INT_FLAG_CH1BRKM</i>	channel 1 BREAK multi-period interrupt flag, <i>TIMERx(x=0,7)</i>

<i>BRKM</i>	
<i>TIMER_INT_FLAG_CH2</i> <i>BRKM</i>	channel 2 BREAK multi-period interrupt flag, $TIMERx(x=0,7)$
<i>TIMER_INT_FLAG_CH0</i> <i>BRK</i>	channel 0 BREAK interrupt flag, $TIMERx(x=0,7)$
<i>TIMER_INT_FLAG_CH1</i> <i>BRK</i>	channel 1 BREAK interrupt flag, $TIMERx(x=0,7)$
<i>TIMER_INT_FLAG_CH2</i> <i>BRK</i>	channel 2 BREAK interrupt flag, $TIMERx(x=0,7)$
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER0 update interrupt flag */
timer_interrupt_flag_clear(TIMER0, TIMER_INT_FLAG_UP);
```

## 3.26. TRIGSEL

TRIGSEL is the trigger selection controller in the MCU. It allows software to select the trigger input signal for various peripherals. The TRIGSEL registers are listed in chapter [3.26.1](#), the TRIGSEL firmware functions are introduced in chapter [3.26.2](#).

### 3.26.1. Descriptions of Peripheral registers

TRIGSEL registers are listed in the table shown as below:

**Table 3-797 TRIGSEL Registers**

Registers	Descriptions
TRIGSEL_EXTOUT_0	TRIGSEL trigger selection for EXTOUT register 0
TRIGSEL_EXTOUT_1	TRIGSEL trigger selection for EXTOUT register 1
TRIGSEL_EXTOUT_2	TRIGSEL trigger selection for EXTOUT register 2
TRIGSEL_EXTOUT_3	TRIGSEL trigger selection for EXTOUT register 3
TRIGSEL_TIMER0ITI	TRIGSEL trigger selection for TIMER0ITI register
TRIGSEL_TIMER1ITI	TRIGSEL trigger selection for TIMER1ITI register
TRIGSEL_TIMER2ITI	TRIGSEL trigger selection for TIMER2ITI register
TRIGSEL_TIMER3ITI	TRIGSEL trigger selection for TIMER3ITI register
TRIGSEL_TIMER4ITI	TRIGSEL trigger selection for TIMER4ITI register
TRIGSEL_TIMER7ITI	TRIGSEL trigger selection for TIMER7ITI register
TRIGSEL_TIMER15ITI	TRIGSEL trigger selection for TIMER15ITI register
TRIGSEL_TIMER16ITI	TRIGSEL trigger selection for TIMER16ITI register
TRIGSEL_DAC	TRIGSEL trigger selection for DAC register

TRIGSEL_ADC0_ROUTRG	TRIGSEL trigger selection for ADC0_ROUTRG register
TRIGSEL_ADC0_INSTRG	TRIGSEL trigger selection for ADC0_INSTRG register
TRIGSEL_ADC1_ROUTRG	TRIGSEL trigger selection for ADC1_ROUTRG register
TRIGSEL_ADC1_INSTRG	TRIGSEL trigger selection for ADC1_INSTRG register
TRIGSEL_ADC2_ROUTRG	TRIGSEL trigger selection for ADC2_ROUTRG register
TRIGSEL_ADC2_INSTRG	TRIGSEL trigger selection for ADC2_INSTRG register
TRIGSEL_TIMER0BRKIN	TRIGSEL trigger selection for TIMER0BRKIN register
TRIGSEL_TIMER0CHBRKIN	TRIGSEL trigger selection for TIMER0CHBRKIN register
TRIGSEL_TIMER7BRKIN	TRIGSEL trigger selection for TIMER7BRKIN register
TRIGSEL_TIMER7CHBRKIN	TRIGSEL trigger selection for TIMER7CHBRKIN register
TRIGSEL_TIMER15BRKIN	TRIGSEL trigger selection for TIMER15BRKIN register
TRIGSEL_TIMER16BRKIN	TRIGSEL trigger selection for TIMER16BRKIN register

### 3.26.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-798. TRIGSEL firmware function**

Function name	Function description
trigsel_deinit	deinitialize TRIGSEL
trigsel_init	set the trigger input signal for target peripheral
trigsel_trigger_source_get	get the trigger input signal for target peripheral
trigsel_register_lock_set	lock the trigger register
trigsel_register_lock_get	get the trigger register lock status

#### Enum trigsel\_source\_enum

**Table 3-799. Enum trigsel\_source\_enum**

Member name	Descriptions
TRIGSEL_INPUT_VSS	trigger input source VSS
TRIGSEL_INPUT_VDD	trigger input source VDD
TRIGSEL_INPUT_TRIGSEL_IN0	trigger input source TRIGSEL_IN0 pin
TRIGSEL_INPUT_TRIGSEL_IN1	trigger input source TRIGSEL_IN1 pin
TRIGSEL_INPUT_TRIGSEL_IN2	trigger input source TRIGSEL_IN2 pin
TRIGSEL_INPUT_TRIGSEL_IN3	trigger input source TRIGSEL_IN3 pin
TRIGSEL_INPUT_TRIGSEL_IN4	trigger input source TRIGSEL_IN4 pin
TRIGSEL_INPUT_TRIGSEL_IN5	trigger input source TRIGSEL_IN5 pin
TRIGSEL_INPUT_TRIGSEL_IN6	trigger input source TRIGSEL_IN6 pin

SEL_IN6	
TRIGSEL_INPUT_TRIG SEL_IN7	trigger input source TRIGSEL_IN7 pin
TRIGSEL_INPUT_TIME R0_TRGO	trigger input source TIMER0_TRGO
TRIGSEL_INPUT_TIME R0_CH0	trigger input source TIMER0_CH0
TRIGSEL_INPUT_TIME R0_CH1	trigger input source TIMER0_CH1
TRIGSEL_INPUT_TIME R0_CH2	trigger input source TIMER0_CH2
TRIGSEL_INPUT_TIME R0_CH3	trigger input source TIMER0_CH3
TRIGSEL_INPUT_TIME R7_TRGO	trigger input source TIMER7_TRGO
TRIGSEL_INPUT_TIME R7_CH0	trigger input source TIMER7_CH0
TRIGSEL_INPUT_TIME R7_CH1	trigger input source TIMER7_CH1
TRIGSEL_INPUT_TIME R7_CH2	trigger input source TIMER7_CH2
TRIGSEL_INPUT_TIME R7_CH3	trigger input source TIMER7_CH3
TRIGSEL_INPUT_TIME R5_TRGO	trigger input source TIMER5_TRGO
TRIGSEL_INPUT_TIME R6_TRGO	trigger input source TIMER6_TRGO
TRIGSEL_INPUT_TIME R1_TRGO	trigger input source TIMER1_TRGO
TRIGSEL_INPUT_TIME R1_CH0	trigger input source TIMER1_CH0
TRIGSEL_INPUT_TIME R1_CH1	trigger input source TIMER1_CH1
TRIGSEL_INPUT_TIME R1_CH2	trigger input source TIMER1_CH2
TRIGSEL_INPUT_TIME R1_CH3	trigger input source TIMER1_CH3
TRIGSEL_INPUT_TIME R2_TRGO	trigger input source TIMER2_TRGO
TRIGSEL_INPUT_TIME R2_CH0	trigger input source TIMER2_CH0
TRIGSEL_INPUT_TIME R2_CH1	trigger input source TIMER2_CH1

TRIGSEL_INPUT_TIME R2_CH2	trigger input source TIMER2_CH2
TRIGSEL_INPUT_TIME R2_CH3	trigger input source TIMER2_CH3
TRIGSEL_INPUT_TIME R3_TRGO	trigger input source TIMER3_TRGO
TRIGSEL_INPUT_TIME R3_CH0	trigger input source TIMER3_CH0
TRIGSEL_INPUT_TIME R3_CH1	trigger input source TIMER3_CH1
TRIGSEL_INPUT_TIME R3_CH2	trigger input source TIMER3_CH2
TRIGSEL_INPUT_TIME R3_CH3	trigger input source TIMER3_CH3
TRIGSEL_INPUT_TIME R4_TRGO	trigger input source TIMER4_TRGO
TRIGSEL_INPUT_TIME R4_CH0	trigger input source TIMER4_CH0
TRIGSEL_INPUT_TIME R4_CH1	trigger input source TIMER4_CH1
TRIGSEL_INPUT_TIME R4_CH2	trigger input source TIMER4_CH2
TRIGSEL_INPUT_TIME R4_CH3	trigger input source TIMER4_CH3
TRIGSEL_INPUT_TIME R15_TRGO	trigger input source TIMER15_TRGO
TRIGSEL_INPUT_TIME R15_CH0	trigger input source TIMER15_CH0
TRIGSEL_INPUT_TIME R15_CH1	trigger input source TIMER15_CH1
TRIGSEL_INPUT_TIME R15_MCH0	trigger input source TIMER15_MCH0
TRIGSEL_INPUT_TIME R16_TRGO	trigger input source TIMER16_TRGO
TRIGSEL_INPUT_TIME R16_CH0	trigger input source TIMER16_CH0
TRIGSEL_INPUT_TIME R16_CH1	trigger input source TIMER16_CH1
TRIGSEL_INPUT_TIME R16_MCH0	trigger input source TIMER16_MCH0
TRIGSEL_INPUT_ADC 0_WD0_OUT	trigger input source ADC0_WD0_OUT
TRIGSEL_INPUT_ADC	trigger input source ADC0_WD1_OUT

0_WD1_OUT	
TRIGSEL_INPUT_ADC 0_WD2_OUT	trigger input source ADC0_WD2_OUT
TRIGSEL_INPUT_ADC 1_WD0_OUT	trigger input source ADC1_WD0_OUT
TRIGSEL_INPUT_ADC 1_WD1_OUT	trigger input source ADC1_WD1_OUT
TRIGSEL_INPUT_ADC 1_WD2_OUT	trigger input source ADC1_WD2_OUT
TRIGSEL_INPUT_ADC 2_WD0_OUT	trigger input source ADC2_WD0_OUT
TRIGSEL_INPUT_ADC 2_WD1_OUT	trigger input source ADC2_WD1_OUT
TRIGSEL_INPUT_ADC 2_WD2_OUT	trigger input source ADC2_WD2_OUT
TRIGSEL_INPUT_CMP 0_OUT	trigger input source CMP0_OUT
TRIGSEL_INPUT_CK_ OUT	trigger input source CK_OUT
TRIGSEL_INPUT_TIME R0_BKIN	trigger input source TIMER0_BKIN
TRIGSEL_INPUT_TIME R0_CH0BKIN	trigger input source TIMER0_CH0BKIN
TRIGSEL_INPUT_TIME R0_CH1BKIN	trigger input source TIMER0_CH1BKIN
TRIGSEL_INPUT_TIME R0_CH2BKIN	trigger input source TIMER0_CH2BKIN
TRIGSEL_INPUT_TIME R7_BKIN	trigger input source TIMER7_BKIN
TRIGSEL_INPUT_TIME R7_CH0BKIN	trigger input source TIMER7_CH0BKIN
TRIGSEL_INPUT_TIME R7_CH1BKIN	trigger input source TIMER7_CH1BKIN
TRIGSEL_INPUT_TIME R7_CH2BKIN	trigger input source TIMER7_CH2BKIN
TRIGSEL_INPUT_TIME R15_BKIN	trigger input source TIMER15_BKIN
TRIGSEL_INPUT_TIME R16_BKIN	trigger input source TIMER16_BKIN
TRIGSEL_INPUT_EXTI 9	trigger input source EXTI9
TRIGSEL_INPUT_EXTI 11	trigger input source EXTI11



TRIGSEL_INPUT_EXTI 15	trigger input source EXTI15
--------------------------	-----------------------------

## Enum trigsel\_periph\_enum

**Table 3-800. Enum trigsel\_periph\_enum**

Member name	Descriptions
TRIGSEL_OUTPUT_T RIGSEL_OUT0	output target peripheral TRIGSEL_OUT0 pin
TRIGSEL_OUTPUT_T RIGSEL_OUT1	output target peripheral TRIGSEL_OUT1 pin
TRIGSEL_OUTPUT_T RIGSEL_OUT2	output target peripheral TRIGSEL_OUT2 pin
TRIGSEL_OUTPUT_T RIGSEL_OUT3	output target peripheral TRIGSEL_OUT3 pin
TRIGSEL_OUTPUT_T RIGSEL_OUT4	output target peripheral TRIGSEL_OUT4 pin
TRIGSEL_OUTPUT_T RIGSEL_OUT5	output target peripheral TRIGSEL_OUT5 pin
TRIGSEL_OUTPUT_T RIGSEL_OUT6	output target peripheral TRIGSEL_OUT6 pin
TRIGSEL_OUTPUT_T RIGSEL_OUT7	output target peripheral TRIGSEL_OUT7 pin
TRIGSEL_OUTPUT_TI MER0_ITI	output target peripheral TIMER0_ITI
TRIGSEL_OUTPUT_TI MER1_ITI	output target peripheral TIMER1_ITI
TRIGSEL_OUTPUT_TI MER2_ITI	output target peripheral TIMER2_ITI
TRIGSEL_OUTPUT_TI MER3_ITI	output target peripheral TIMER3_ITI
TRIGSEL_OUTPUT_TI MER4_ITI	output target peripheral TIMER4_ITI
TRIGSEL_OUTPUT_TI MER7_ITI	output target peripheral TIMER7_ITI
TRIGSEL_OUTPUT_TI MER15_ITI	output target peripheral TIMER15_ITI
TRIGSEL_OUTPUT_TI MER16_ITI	output target peripheral TIMER16_ITI
TRIGSEL_OUTPUT_D AC_OUT_EXTRG	output target peripheral DAC_OUT_EXTRG
TRIGSEL_OUTPUT_A DC0_ROUTRG	output target peripheral ADC0_ROUTRG
TRIGSEL_OUTPUT_A	output target peripheral ADC0_INSTRG

DC0_INSTRG	
TRIGSEL_OUTPUT_A DC1_ROUTRG	output target peripheral ADC1_ROUTRG
TRIGSEL_OUTPUT_A DC1_INSTRG	output target peripheral ADC1_INSTRG
TRIGSEL_OUTPUT_A DC2_ROUTRG	output target peripheral ADC2_ROUTRG
TRIGSEL_OUTPUT_A DC2_INSTRG	output target peripheral ADC2_INSTRG
TRIGSEL_OUTPUT_TI MER0_BRKIN	output target peripheral TIMER0_BRKIN
TRIGSEL_OUTPUT_TI MER0_CH0BRKIN	output target peripheral TIMER0_CH0BRKIN
TRIGSEL_OUTPUT_TI MER0_CH1BRKIN	output target peripheral TIMER0_CH1BRKIN
TRIGSEL_OUTPUT_TI MER0_CH2BRKIN	output target peripheral TIMER0_CH2BRKIN
TRIGSEL_OUTPUT_TI MER7_BRKIN	output target peripheral TIMER7_BRKIN
TRIGSEL_OUTPUT_TI MER7_CH0BRKIN	output target peripheral TIMER7_CH0BRKIN
TRIGSEL_OUTPUT_TI MER7_CH1BRKIN	output target peripheral TIMER7_CH1BRKIN
TRIGSEL_OUTPUT_TI MER7_CH2BRKIN	output target peripheral TIMER7_CH2BRKIN
TRIGSEL_OUTPUT_TI MER15_BRKIN	output target peripheral TIMER15_BRKIN
TRIGSEL_OUTPUT_TI MER16_BRKIN	output target peripheral TIMER16_BRKIN

## trigsel\_deinit

The description of trigsel\_deinit is shown as below:

**Table 3-801. Function trigsel\_deinit**

<b>Function name</b>	trigsel_deinit
<b>Function prototype</b>	void trigsel_deinit(void)
<b>Function descriptions</b>	deinitialize TRIGSEL (API_ID(0x0001U))
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* deinitialize TRIGSEL (API_ID(0x0001U)) */
trigsel_deinit();
```

## trigsel\_init

The description of trigsel\_init is shown as below:

**Table 3-802. Function trigsel\_init**

Function name	trigsel_init
Function prototype	void trigsel_init(trigsel_periph_enum target_periph, trigsel_source_enum trigger_source)
Function descriptions	set the trigger input signal for target peripheral (API_ID(0x0002U))
Precondition	-
The called functions	-
Input parameter{in}	
target_periph	Target peripheral value, refer to <a href="#">Table 3-800. Enum trigsel_periph_enum</a>
Input parameter{in}	
trigger_source	Trigger source value, refer to <a href="#">Table 3-799. Enum trigsel_source_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the trigger input signal for target peripheral (API_ID(0x0002U)) */
trigsel_init(TRIGSEL_OUTPUT_TIMER1_ITI, TRIGSEL_INPUT_CK_OUT);;
```

## trigsel\_trigger\_source\_get

The description of trigsel\_trigger\_source\_get is shown as below:

**Table 3-803. Function trigsel\_trigger\_source\_get**

Function name	trigsel_trigger_source_get
Function prototype	trigsel_source_enum trigsel_trigger_source_get(trigsel_periph_enum target_periph)
Function descriptions	get the trigger input signal for target peripheral (API_ID(0x0003U))
Precondition	-
The called functions	-
Input parameter{in}	
target_periph	Target peripheral value, refer to <a href="#">Table 3-800. Enum trigsel_periph_enum</a>
Output parameter{out}	

-	-
<b>Return value</b>	
<b>trigger_source</b>	trigger source value(0~0x4E)

Example:

```
/* get the trigger input signal for target peripheral (API_ID(0x0003U)) */
trigsel_trigger_source_get(TRIGSEL_OUTPUT_TIMER16_BRKIN);
```

### trigsel\_register\_lock\_set

The description of trigsel\_register\_lock\_set is shown as below:

**Table 3-804.Function trigsel\_register\_lock\_set**

<b>Function name</b>	trigsel_register_lock_set
<b>Function prototype</b>	void trigsel_register_lock_set(trigsel_periph_enum target_periph)
<b>Function descriptions</b>	lock the trigger register (API_ID(0x0004U))
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>target_periph</b>	Target peripheral value, refer to <a href="#">Table 3-800. Enum trigsel_periph_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock the trigger register (API_ID(0x0004U)) */
trigsel_register_lock_set(TRIGSEL_OUTPUT_TIMER16_BRKIN);
```

### trigsel\_register\_lock\_get

The description of trigsel\_register\_lock\_get is shown as below:

**Table 3-805.Function trigsel\_register\_lock\_get**

<b>Function name</b>	trigsel_register_lock_get
<b>Function prototype</b>	FlagStatus trigsel_register_lock_get(trigsel_periph_enum target_periph)
<b>Function descriptions</b>	
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>target_periph</b>	Target peripheral value, refer to <a href="#">Table 3-800. Enum trigsel_periph_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>flag</b>	SET or RESET

Example:

```
/*get the trigger register lock status*/
```

```
trngsel_register_lock_get(TRNGSEL_OUTPUT_TIMER16_BRKIN);
```

## 3.27. TRNG

The true random number generator (TRNG) module can generate a 32-bit value using continuous analog noise. The TRNG registers are listed in chapter [3.27.1](#). The TRNG firmware functions are introduced in chapter [3.27.2](#).

### 3.27.1. Descriptions of Peripheral registers

TRNG registers are listed in the table shown as below:

**Table 3-806 TRNG Registers**

Registers	Descriptions
TRNG_CTL	TRNG control register
TRNG_STAT	TRNG status register
TRNG_DATA	TRNG data register

### 3.27.2. Descriptions of Peripheral functions

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

**Table 3-807. TRNG firmware function**

Function name	Function description
trng_deinit	deinitialize the TRNG
trng_enable	enable the TRNG interface
trng_disable	disable the TRNG interface
trng_get_true_random_data	get the true random data
trng_powermode_config	configure TRNG analog power mode
trng_flag_get	get the TRNG status flags
trng_interrupt_enable	enable the TRNG interrupt
trng_interrupt_disable	disable the TRNG interrupt
trng_interrupt_flag_get	get the TRNG interrupt flags
trng_interrupt_flag_clear	clear the TRNG interrupt flags

#### Enum trng\_flag\_enum

**Table 3-808. Enum trng\_flag\_enum**

Member name	Function description
TRNG_FLAG_DRDY	random data ready status
TRNG_FLAG_CECS	clock error current status
TRNG_FLAG_SECS	seed error current status

## Enum trng\_int\_flag\_enum

**Table 3-809. Enum trng\_int\_flag\_enum**

Member name	Function description
TRNG_INT_FLAG_CEIF	clock error interrupt flag
TRNG_INT_FLAG_SEIF	seed error interrupt flag

## trng\_deinit

The description of trng\_deinit is shown as below:

**Table 3-810. Function trng\_deinit**

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

Example:

```
/* reset TRNG */
trng_deinit();
```

## trng\_enable

The description of trng\_enable is shown as below:

**Table 3-811. Function trng\_enable**

Function name	trng_enable
Function prototype	void trng_enable(void)
Function descriptions	enable the TRNG interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TRNG */
trng_enable();
```

### trng\_disable

The description of trng\_disable is shown as below:

**Table 3-812 Function trng\_disable**

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

Example:

```
/* disable TRNG */
trng_disable();
```

### trng\_get\_true\_random\_data

The description of trng\_get\_true\_random\_data is shown as below:

**Table 3-813 Function trng\_get\_true\_random\_data**

<b>Function name</b>	trng_get_true_random_data
<b>Function prototype</b>	uint32_t trng_get_true_random_data(void)
<b>Function descriptions</b>	get the true random data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	the generated random data, 0x0 – 0xFFFFFFFF

Example:

```
/* get true random data */
```

```
uint32_t data;

data = trng_get_true_random_data();
```

### trng\_powermode\_config

The description of trng\_powermode\_config is shown as below:

**Table 3-814 trng\_flag\_get**

<b>Function name</b>	trng_powermode_config
<b>Function prototype</b>	void trng_powermode_config(uint32_t powermode);
<b>Function descriptions</b>	configure TRNG analog power mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>powermode</b>	the power mode selection
TRNG_NR_ULTRALOW	TRNG analog power mode ultralow
TRNG_NR_LOW	TRNG analog power mode low
TRNG_NR_MEDIUM	TRNG analog power mode medium
TRNG_NR_HIGH	TRNG analog power mode high
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TRNG analog power mode */

trng_powermode_config (TRNG_FLAG_CECS);
```

### trng\_flag\_get

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

**Table 3-815 trng\_flag\_get**

<b>Function name</b>	trng_flag_get
<b>Function prototype</b>	FlagStatus trng_flag_get(trng_flag_enum flag);
<b>Function descriptions</b>	get the trng status flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	trng_flag_enum, please refer to <a href="#">Table 3-808. Enum trng_flag_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	



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

Example:

```
/* get TRNG clock error current flag status */

FlagStatus flag_status = RESET;

flag_status == trng_flag_get(TRNG_FLAG_CECS);
```

### trng\_interrupt\_enable

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

**Table 3-816 trng\_interrupt\_enable**

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

Example:

```
/* enable TRNG interrupt */

trng_interrupt_enable();
```

### trng\_interrupt\_disable

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

**Table 3-817 trng\_interrupt\_disable**

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

Example:

```
/* disable TRNG interrupt */
trng_interrupt_disable();
```

### trng\_interrupt\_flag\_get

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

**Table 3-818 trng\_interrupt\_flag\_get**

<b>Function name</b>	trng_interrupt_flag_get
<b>Function prototype</b>	FlagStatus trng_interrupt_flag_get(trng_int_flag_enum int_flag);
<b>Function descriptions</b>	get the trng interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	trng_flag_enum, please refer to <a href="#">Table 3-809. Enum trng_int_flag_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TRNG clock error interrupt flag */
FlagStatus interrupt_flag = RESET;
interrupt_flag = trng_interrupt_flag_get(TRNG_INT_FLAG_CEIF);
```

### trng\_interrupt\_flag\_clear

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

**Table 3-819 trng\_interrupt\_flag\_clear**

<b>Function name</b>	trng_interrupt_flag_clear
<b>Function prototype</b>	void trng_interrupt_flag_get(trng_int_flag_enum int_flag);
<b>Function descriptions</b>	clear the trng interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	trng_flag_enum, please refer to <a href="#">Table 3-809. Enum trng_int_flag_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* clear TRNG clock error interrupt flag */

trng_interrupt_flag_clear(TRNG_INT_FLAG_CEIF);

```

## 3.28. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.28.1](#), the USART firmware functions are introduced in chapter [3.28.2](#).

### 3.28.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

**Table 3-820. 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
USART_CHC	Coherence control register

### 3.28.2. Descriptions of Peripheral functions

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

**Table 3-821. USART firmware function**

Function name	Function description
usart_deinit	reset USART/UART
usart_baudrate_set	configure USART baud rate value
usart_parity_config	configure USART parity
usart_word_length_set	configure USART word length
usart_stop_bit_set	configure USART stop bit length
usart_enable	enable USART
usart_disable	disable USART

Function name	Function description
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_oversample_config	configure the USART oversample mode
usart_sample_bit_config	configure sample bit method
usart_receiver_timeout_enable	enable receiver timeout
usart_receiver_timeout_disable	disable receiver timeout
usart_receiver_timeout_threshold_config	configure receiver timeout threshold
usart_data_transmit	USART transmit data function
usart_data_receive	USART receive data function
usart_address_config	configure 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
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_break_frame_coherence_config	configure break frame coherence mode

Function name	Function description
usart_parity_check_coherence_config	configure parity check coherence mode
usart_hardware_flow_coherence_config	configure hardware flow control coherence mode
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

### Enum usart\_flag\_enum

Table 3-822. Enum usart\_flag\_enum

Member name	Function description
USART_FLAG_CTS	CTS change flag
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_TBE	transmit data buffer empty
USART_FLAG_RBNE	transmission complete
USART_FLAG_TC	read data buffer not empty
USART_FLAG_IDLE	IDLE frame detected flag
USART_FLAG_ORERR	overrun error
USART_FLAG_NERR	noise error flag
USART_FLAG_FERR	frame error flag
USART_FLAG_PERR	parity error flag
USART_FLAG_BSY	busy flag
USART_FLAG_EB	end of block flag
USART_FLAG_RT	receiver timeout flag
USART_FLAG_EPERR	early parity error flag

### Enum usart\_interrupt\_flag\_enum

Table 3-823. Enum usart\_interrupt\_flag\_enum

Member name	Function description
USART_INT_FLAG_PERR	parity error interrupt and flag
USART_INT_FLAG_TBE	transmitter buffer empty interrupt and flag
USART_INT_FLAG_TC	transmission complete interrupt and flag
USART_INT_FLAG_RBNE	read data buffer not empty interrupt and flag
USART_INT_FLAG_RBNE_ORERR	read data buffer not empty interrupt and overrun error flag
USART_INT_FLAG_IDLE	IDLE line detected interrupt and flag
USART_INT_FLAG_LBD	LIN break detected interrupt and flag

USART_INT_FLAG_CTS	CTS interrupt and flag
USART_INT_FLAG_ERR_ORER R	error interrupt and overrun error
USART_INT_FLAG_ERR_NERR	error interrupt and noise error flag
USART_INT_FLAG_ERR_FERR	error interrupt and frame error flag
USART_INT_FLAG_EB	interrupt enable bit of end of block event and flag
USART_INT_FLAG_RT	interrupt enable bit of receive timeout event and flag

## Enum usart\_interrupt\_enum

**Table 3-824. Enum usart\_interrupt\_enum**

Member name	Function description
USART_INT_PERR	parity error interrupt
USART_INT_TBE	transmitter buffer empty interrupt
USART_INT_TC	transmission complete interrupt
USART_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt
USART_INT_IDLE	IDLE line detected interrupt
USART_INT_LBD	LIN break detected interrupt
USART_INT_CTS	CTS interrupt
USART_INT_ERR	error interrupt
USART_INT_EB	interrupt enable bit of end of block event
USART_INT_RT	interrupt enable bit of receive timeout event

## Enum usart\_invert\_enum

**Table 3-825. Enum usart\_invert\_enum**

Member name	Function description
USART_DINV_ENABLE	data bit level inversion
USART_DINV_DISABLE	data bit level not inversion
USART_TXPIN_ENABLE	TX pin level inversion
USART_TXPIN_DISABLE	TX pin level not inversion
USART_RXPIN_ENABLE	RX pin level inversion
USART_RXPIN_DISABLE	RX pin level not inversion

## usart\_deinit

The description of usart\_deinit is shown as below:

**Table 3-826. Function usart\_deinit**

Function name	usart_deinit
Function prototype	void usart_deinit(uint32_t usart_periph);
Function descriptions	reset USART/UART
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	

<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 3-827. Function usart\_baudrate\_set**

<b>Function name</b>	usart_baudrate_set
<b>Function prototype</b>	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
<b>Function descriptions</b>	configure USART baud rate value
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/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 3-828. 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 3-829. 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 3-830. 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
USARTx	x=0,1,2
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>stblen</b>	USART stop bit
USART_STB_1BIT	1 bit
USART_STB_0_5BIT	0.5 bit, not available for UARTx(x=3,4)
USART_STB_2BIT	2 bits
USART_STB_1_5BIT	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 3-831. Function usart\_enable**

<b>Function name</b>	usart_enable
<b>Function prototype</b>	void usart_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable USART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>usart_periph</b>	USARTx/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 */
```

```
usart_enable(USART0);
```

### usart\_disable

The description of usart\_disable is shown as below:

**Table 3-832. Function usart\_disable**

<b>Function name</b>	usart_disable
<b>Function prototype</b>	void usart_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable USART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/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 */
```

```
usart_disable(USART0);
```

### usart\_transmit\_config

The description of usart\_transmit\_config is shown as below:

**Table 3-833. 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>	disable USART transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 transmitter */
```

```
usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

### usart\_receive\_config

The description of usart\_receive\_config is shown as below:

**Table 3-834. 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 3-835. 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 3-836. 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

Input parameter{in}	
invertpara	refer to enum usart_invert_enum
USART_DINV_ENABLER	data bit level inversion
USART_DINV_DISABLER	data bit level not inversion
USART_TXPIN_ENABLER	TX pin level inversion
USART_TXPIN_DISABLER	TX pin level not inversion
USART_RXPIN_ENABLER	RX pin level inversion
USART_RXPIN_DISABLER	RX pin level not inversion
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART inversion */
```

```
usart_invert_config(USART0, USART_DINV_ENABLE);
```

### usart\_oversample\_config

The description of usart\_oversample\_config is shown as below:

**Table 3-837. Function usart\_oversample\_config**

Function name	usart_oversample_config
Function prototype	void usart_oversample_config(uint32_t usart_periph, uint32_t oversamp);
Function descriptions	configure the USART oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Input parameter{in}	
oversamp	oversample value
USART_OVSMOD_8	8 bits
USART_OVSMOD_16	16 bits
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure USART0 oversample mode */
usart_oversample_config(USART0, USART_OVSMOD_8);
```

### usart\_sample\_bit\_config

The description of usart\_sample\_bit\_config is shown as below:

**Table 3-838. Function usart\_sample\_bit\_config**

<b>Function name</b>	usart_sample_bit_config
<b>Function prototype</b>	void usart_sample_bit_config(uint32_t usart_periph, uint32_t obsm);
<b>Function descriptions</b>	configure sample bit method
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>obsm</b>	sample bit
USART_OSB_1bit	1 bits
USART_OSB_3bit	3 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 sample bit */
usart_sample_bit_config(USART0, USART_OSB_1bit);
```

### usart\_receiver\_timeout\_enable

The description of usart\_receiver\_timeout\_enable is shown as below:

**Table 3-839. Function usart\_receiver\_timeout\_enable**

<b>Function name</b>	usart_receiver_timeout_enable
<b>Function prototype</b>	void usart_receiver_timeout_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable receiver timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable receiver timeout of USART */
usart_receiver_timeout_enable(USART0);
```

### usart\_receiver\_timeout\_disable

The description of usart\_receiver\_timeout\_disable is shown as below:

**Table 3-840. Function usart\_receiver\_timeout\_disable**

<b>Function name</b>	usart_receiver_timeout_disable
<b>Function prototype</b>	void usart_receiver_timeout_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable receiver timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable receiver timeout of USART */
usart_receiver_timeout_disable(USART0);
```

### usart\_receiver\_timeout\_threshold\_config

The description of usart\_receiver\_timeout\_threshold\_config is shown as below:

**Table 3-841. 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>	-

Input parameter{in}	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
Input parameter{in}	
<b>rtimeout</b>	timeout value
<i>0-0xFFFFF</i>	timeout value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the receiver timeout threshold of USART0 */
```

```
usart_receiver_timeout_threshold_config(USART0, 115200*3);
```

### usart\_data\_transmit

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

**Table 3-842. 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>	-
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>data</b>	data of transmission
<i>0-0xFF</i>	data of transmission
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 transmit data */
```

```
usart_data_transmit(USART0, 0xAA);
```

### usart\_data\_receive

The description of usart\_data\_receive is shown as below:



Table 3-843. Function `usart_data_receive`

Function name	<code>usart_data_receive</code>
Function prototype	<code>uint16_t usart_data_receive(uint32_t usart_periph);</code>
Function descriptions	USART receive data function
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	x=0,1,2
<code>UARTx</code>	x=3,4
Output parameter{out}	
-	-
Return value	
<code>uint32_t</code>	data of received(0-0xFF)

Example:

```
/* USART0 receive data */
```

```
uint16_t temp;
```

```
temp = usart_data_receive(USART0);
```

### `usart_address_config`

The description of `usart_address_config` is shown as below:

Table 3-844. Function `usart_address_config`

Function name	<code>usart_address_config</code>
Function prototype	<code>void usart_address_config(uint32_t usart_periph, uint8_t addr);</code>
Function descriptions	configure the address of the USART in wake up by address match mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>usart_periph</code>	USARTx/UARTx peripheral
<code>USARTx</code>	x=0,1,2
<code>UARTx</code>	x=3,4
Input parameter{in}	
<code>addr</code>	address of USART/UART
<code>0-0xFF</code>	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 3-845. Function usart\_mute\_mode\_enable**

<b>Function name</b>	usart_mute_mode_enable
<b>Function prototype</b>	void usart_mute_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	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 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 3-846. 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 3-847. Function usart\_mute\_mode\_wakeup\_config**

<b>Function name</b>	usart_mute_mode_wakeup_config
<b>Function prototype</b>	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
<b>Function descriptions</b>	configure wakeup method in mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>wmethod</b>	two methods be used to enter or exit the mute mode
USART_WM_IDLE	idle line
USART_WM_ADDR	address mask
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 wakeup method in mute mode */
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

### usart\_lin\_mode\_enable

The description of usart\_lin\_mode\_enable is shown as below:

**Table 3-848. Function usart\_lin\_mode\_enable**

<b>Function name</b>	usart_lin_mode_enable
<b>Function prototype</b>	void usart_lin_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable LIN mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral

USARTx	x=0,1,2
UARTx	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 3-849. Function usart\_lin\_mode\_disable**

Function name	usart_lin_mode_disable
Function prototype	void usart_lin_mode_disable(uint32_t usart_periph);
Function descriptions	disable LIN mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 LIN mode disable */
```

```
usart_lin_mode_disable(USART0);
```

### usart\_lin\_break\_detection\_length\_config

The description of usart\_lin\_break\_detection\_length\_config is shown as below:

**Table 3-850. Function usart\_lin\_break\_detection\_length\_config**

Function name	usart_lin_break_detection_length_config
Function prototype	void usart_lin_break_detection_length_config(uint32_t usart_periph, uint32_t lflen);
Function descriptions	configure LIN break frame length
Precondition	-

The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Input parameter{in}	
lblen	lin break frame length
USART_LBLEN_10B	break frame length is 10 bits
USART_LBLEN_11B	break frame length is 11 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure LIN break frame length */
```

```
usart_lin_break_detection_length_config(USART0, USART_LBLEN_10B);
```

### usart\_send\_break

The description of usart\_send\_break is shown as below:

**Table 3-851. Function usart\_send\_break**

Function name	usart_send_break
Function prototype	void usart_send_break(uint32_t usart_periph);
Function descriptions	send break frame
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 send break frame */
```

```
usart_send_break(USART0);
```

### usart\_halfduplex\_enable

The description of usart\_halfduplex\_enable is shown as below:

**Table 3-852. 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 3-853. 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 3-854. 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 3-855. 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 3-856. 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
USARTx	x=0,1,2
<b>Input parameter{in}</b>	
<b>clen</b>	CK length
USART_CLEN_NONE	there are 7 CK pulses for an 8 bit frame and 8 CK pulses for a 9 bit frame
USART_CLEN_EN	there are 8 CK pulses for an 8 bit frame and 9 CK pulses for a 9 bit frame
<b>Input parameter{in}</b>	
<b>cph</b>	clock phase
USART_CPH_1CK	first clock transition is the first data capture edge
USART_CPH_2CK	second clock transition is the first data capture edge
<b>Input parameter{in}</b>	
<b>cpl</b>	clock polarity
USART_CPL_LOW	steady low value on CK pin
USART_CPL_HIGH	steady high value on CK pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 synchronous mode parameters */

usart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,
USART_CPL_HIGH);
```

## usart\_guard\_time\_config

The description of usart\_guard\_time\_config is shown as below:

**Table 3-857. 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
<i>0-0x000000FF</i>	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 3-858. Function usart\_smartcard\_mode\_enable**

<b>Function name</b>	usart_smartcard_mode_enable
<b>Function prototype</b>	void usart_smartcard_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 smartcard mode enable */
```

```
usart_smartcard_mode_enable(USART0);
```

### usart\_smartcard\_mode\_disable

The description of usart\_smartcard\_mode\_disable is shown as below:

**Table 3-859. Function usart\_smartcard\_mode\_disable**

<b>Function name</b>	usart_smartcard_mode_disable
<b>Function prototype</b>	void usart_smartcard_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 smartcard mode disable */
usart_smartcard_mode_disable(USART0);
```

### usart\_smartcard\_mode\_nack\_enable

The description of usart\_smartcard\_mode\_nack\_enable is shown as below:

**Table 3-860. 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 3-861. 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 3-862. 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
<i>0-0xFFFFFFFF</i>	smartcard auto-retry number
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-863. 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 in Smartcard T=1 reception
<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>bl</b>	block length
<i>0-0xFFFFFFFF</i>	block length
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-864. 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
<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 IrDA mode */
```

```
usart_irda_mode_enable(USART0);
```

### usart\_irda\_mode\_disable

The description of usart\_irda\_mode\_disable is shown as below:

**Table 3-865. Function usart\_irda\_mode\_disable**

<b>Function name</b>	usart_irda_mode_disable
<b>Function prototype</b>	void usart_irda_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable IrDA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 IrDA mode */
```

```
usart_irda_mode_disable(USART0);
```

### usart\_prescaler\_config

The description of usart\_prescaler\_config is shown as below:

**Table 3-866. Function usart\_prescaler\_config**

<b>Function name</b>	usart_prescaler_config
<b>Function prototype</b>	void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);
<b>Function descriptions</b>	configure the peripheral clock prescaler in USART IrDA low-power mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
<b>Input parameter{in}</b>	
<b>psc</b>	0x00-0xFF
<b>Output parameter{out}</b>	
-	-

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 3-867. Function usart\_irda\_lowpower\_config**

Function name	usart_irda_lowpower_config
Function prototype	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
Function descriptions	configure IrDA low-power
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2
UARTx	x=3,4
Input parameter{in}	
irlp	IrDA low-power or normal
USART_IRLP_LOW	low-power
USART_IRLP_NORMAL	normal
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

### usart\_hardware\_flow\_rts\_config

The description of usart\_hardware\_flow\_rts\_config is shown as below:

**Table 3-868. Function usart\_hardware\_flow\_rts\_config**

Function name	usart_hardware_flow_rts_config
Function prototype	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
Function descriptions	configure hardware flow control RTS

<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 3-869. Function usart\_hardware\_flow\_cts\_config**

<b>Function name</b>	usart_hardware_flow_cts_config
<b>Function prototype</b>	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
<b>Function descriptions</b>	configure hardware flow control CTS
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	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\_break\_frame\_coherence\_config

The description of usart\_break\_frame\_coherence\_config is shown as below:

**Table 3-870. Function usart\_break\_frame\_coherence\_config**

<b>Function name</b>	usart_break_frame_coherence_config
<b>Function prototype</b>	void usart_break_frame_coherence_config(uint32_t usart_periph, uint32_t bcm);
<b>Function descriptions</b>	configure break frame coherence 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>bcm</b>	
<i>USART_BCM_NONE</i>	No parity error is detected
<i>USART_BCM_EN</i>	Parity error is detected
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 break frame coherence mode */
```

```
usart_break_frame_coherence_config(USART0, USART_BCM_NONE);
```

### usart\_parity\_check\_coherence\_config

The description of usart\_parity\_check\_coherence\_config is shown as below:

**Table 3-871. Function usart\_parity\_check\_coherence\_config**

<b>Function name</b>	usart_parity_check_coherence_config
<b>Function prototype</b>	void usart_parity_check_coherence_config(uint32_t usart_periph, uint32_t pcm);
<b>Function descriptions</b>	configure parity check coherence 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>pcm</b>	
<i>USART_PCM_NONE</i>	not check parity
<i>USART_PCM_EN</i>	check the parity
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 parity check coherence mode */
```

```
usart_parity_check_coherence_config(USART0, USART_PCM_NONE);
```

### usart\_hardware\_flow\_coherence\_config

The description of usart\_hardware\_flow\_coherence\_config is shown as below:

**Table 3-872. Function usart\_hardware\_flow\_coherence\_config**

<b>Function name</b>	usart_hardware_flow_coherence_config
<b>Function prototype</b>	void usart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);
<b>Function descriptions</b>	configure hardware flow control coherence 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>hcm</b>	
<i>USART_HCM_NONE</i>	nRTS signal equals to the rxne status register
<i>USART_HCM_EN</i>	nRTS signal is set when the last data bit has been sampled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure hardware flow control coherence mode */
```

```
usart_hardware_flow_coherence_config(USART0, USART_HCM_NONE);
```

### usart\_dma\_receive\_config

The description of usart\_dma\_receive\_config is shown as below:

Table 3-873. Function `usart_dma_receive_config`

<b>Function name</b>	<code>usart_dma_receive_config</code>
<b>Function prototype</b>	<code>void usart_dma_receive_config(uint32_t usart_periph, uint32_t dmacmd);</code>
<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=3,4
<b>Input parameter{in}</b>	
<b>dmacmd</b>	enable or disable DMA for reception
<i>USART_DENR_ENAB</i> <i>E</i>	DMA enable for reception
<i>USART_DENR_DISAB</i> <i>LE</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_DENR_ENABLE);
```

### `usart_dma_transmit_config`

The description of `usart_dma_transmit_config` is shown as below:

Table 3-874. Function `usart_dma_transmit_config`

<b>Function name</b>	<code>usart_dma_transmit_config</code>
<b>Function prototype</b>	<code>void usart_dma_transmit_config(uint32_t usart_periph, uint32_t dmacmd);</code>
<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=3,4
<b>Input parameter{in}</b>	
<b>dmacmd</b>	enable or disable DMA for transmission
<i>USART_DENT_ENAB</i> <i>E</i>	DMA enable for transmission
<i>USART_DENT_DISAB</i>	DMA disable for transmission

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

Example:

```
/* USART0 DMA enable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_DENT_ENABLE);
```

### usart\_flag\_get

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

**Table 3-875. 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/CHC register
<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>flag</b>	USART flags, refer to <a href="#">Enum usart_flag_enum</a>
Output parameter{out}	
-	-
Return value	
<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 3-876. 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 <a href="#">Enum usart_flag_enum</a>
<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
<i>USART_FLAG_EPERR</i>	early parity error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear USART0 flag */
```

```
usart_flag_clear(USART0,USART_FLAG_TC);
```

## usart\_interrupt\_enable

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

**Table 3-877. 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, refer to <a href="#">Enum usart_interrupt_enum</a>
<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 3-878. 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>interrupt</b>	USART interrupt, refer to <a href="#">Enum usart_interrupt_enum</a>
<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 3-879. Function usart\_interrupt\_flag\_get**

<b>Function name</b>	usart_interrupt_flag_get
<b>Function prototype</b>	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	get USART interrupt and 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 <a href="#">Enum usart_interrupt_flag_enum</a>
<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 3-880. 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 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>int_flag</b>	USART interrupt flag, refer to <a href="#">Enum usart_interrupt_flag_enum</a>
<i>USART_INT_FLAG_CTS</i>	CTS change flag
<i>USART_INT_FLAG_LBD</i>	LIN break detected flag
<i>USART_INT_FLAG_TC</i>	transmission complete
<i>USART_INT_FLAG_RBNE</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>	
-	-

Return value	
-	-

Example:

```
/* clear the USART0 interrupt flag bit status */
```

```
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

## 3.29. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.29.1](#), the WWDGT firmware functions are introduced in chapter [3.29.2](#).

### 3.29.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

**Table 3-881. WWDGT Registers**

Registers	Descriptions
WWDGT_CTL	Control register
WWDGT_CFG	Configuration register
WWDGT_STAT	Status register

### 3.29.2. Descriptions of Peripheral functions

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

**Table 3-882. WWDGT firmware function**

Function name	Function description
wwdgt_deinit	reset the window watchdog timer configuration
wwdgt_enable	start the window watchdog timer counter
wwdgt_counter_update	configure the window watchdog timer counter value
wwdgt_config	configure counter value, window value, and prescaler divider value
wwdgt_flag_get	check early wakeup interrupt state of WWDGT
wwdgt_flag_clear	clear early wakeup interrupt state of WWDGT
wwdgt_interrupt_enable	enable early wakeup interrupt of WWDGT

#### wwdgt\_deinit

The description of wwdgt\_deinit is shown as below:

**Table 3-883. Function wwdgt\_deinit**

Function name	wwdgt_deinit
---------------	--------------

<b>Function prototype</b>	void wwdgt_deinit(void);
<b>Function descriptions</b>	reset the window watchdog timer configuration
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset the window watchdog timer configuration */
```

```
wwdgt_deinit();
```

### wwdgt\_enable

The description of wwdgt\_enable is shown as below:

**Table 3-884. 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 3-885. 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>	
<b>counter_value</b>	0x00 - 0x7F
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* update WWDGT counter to 0x7F */
```

```
wwdgt_counter_update(127);
```

### wwdgt\_config

The description of wwdgt\_config is shown as below:

**Table 3-886. 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
WWDGT_CFG_PSC_D IV16	the time base of window watchdog counter = (PCLK1 / 4096) / 16
WWDGT_CFG_PSC_D IV32	the time base of window watchdog counter = (PCLK1 / 4096) / 32
WWDGT_CFG_PSC_D IV64	the time base of window watchdog counter = (PCLK1 / 4096) / 64
WWDGT_CFG_PSC_D	the time base of window watchdog counter = (PCLK1 / 4096) / 128

IV128	
WWDGT_CFG_PSC_D IV256	the time base of window watchdog counter = (PCLK1 / 4096) / 256
WWDGT_CFG_PSC_D IV512	the time base of window watchdog counter = (PCLK1 / 4096) / 512
WWDGT_CFG_PSC_D IV1024	the time base of window watchdog counter = (PCLK1 / 4096) / 1024
WWDGT_CFG_PSC_D IV2048	the time base of window watchdog counter = (PCLK1 / 4096) / 2048
WWDGT_CFG_PSC_D IV4096	the time base of window watchdog counter = (PCLK1 / 4096) / 4096
WWDGT_CFG_PSC_D IV8192	the time base of window watchdog counter = (PCLK1 / 4096) / 8192
WWDGT_CFG_PSC_D IV16384	the time base of window watchdog counter = (PCLK1 / 4096) / 16384
WWDGT_CFG_PSC_D IV32768	the time base of window watchdog counter = (PCLK1 / 4096) / 32768
WWDGT_CFG_PSC_D IV65536	the time base of window watchdog counter = (PCLK1 / 4096) / 65536
WWDGT_CFG_PSC_D IV131072	the time base of window watchdog counter = (PCLK1 / 4096) / 131072
WWDGT_CFG_PSC_D IV262144	the time base of window watchdog counter = (PCLK1 / 4096) / 262144
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

### wwdgt\_flag\_get

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

**Table 3-887. Function wwdgt\_flag\_get**

<b>Function name</b>	wwdgt_flag_get
<b>Function prototype</b>	FlagStatus wwdgt_flag_get(void);
<b>Function descriptions</b>	check early wakeup interrupt state of WWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get( );
```

```
if(status == RESET)
```

```
{
```

```
...
```

```
}else
```

```
{
```

```
...
```

```
}
```

### wwdgt\_flag\_clear

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

**Table 3-888. 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>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear early wakeup interrupt state of WWDGT */
```

```
wwdgt_flag_clear();
```

## wwdgt\_interrupt\_enable

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

**Table 3-889. Function wwdgt\_interrupt\_enable**

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

Example:

```
/* enable early wakeup interrupt of WWDGT */
wwdgt_interrupt_enable();
```

## 4. Revision history

**Table 4-1. Revision history**

Revision No.	Description	Date
1.0	1. Initial Release	Nov.03, 2025
1.1	1. Input parameter modification: dac_flag_get dac_flag_clear dac_interrupt_enable dac_interrupt_disable dac_interrupt_flag_get dac_interrupt_flag_clear 2. Add can transmission stop function return value description	Feb.25, 2026

## Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company according to the laws of the People's Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively "Unintended Uses"). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby release the Company as well as its suppliers and/or distributors from, any claim, damage, or other liability arising from or related to all Unintended Uses of the Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.