

**GigaDevice Semiconductor Inc.**

**GD32A490**

**ARM® Cortex™-M4 32-bit MCU**

**Firmware Library  
User Guide**

Revision 1.1

(May. 2025)

# 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>27</b>
<b>1.1. Rules of User Manual and Firmware Library .....</b>	<b>27</b>
1.1.1. Peripherals.....	27
1.1.2. Naming rules.....	28
<b>2. Firmware Library Overview.....</b>	<b>30</b>
<b>2.1. File Structure of Firmware Library .....</b>	<b>30</b>
2.1.1. Examples Folder .....	31
2.1.2. Firmware Folder.....	31
2.1.3. Project Folder.....	31
2.1.4. Template Folder .....	31
2.1.5. Utilities Folder .....	34
<b>2.2. File descriptions of Firmware Library .....</b>	<b>35</b>
<b>3. Firmware Library of Standard Peripherals .....</b>	<b>36</b>
<b>3.1. Overview of Firmware Library of Standard Peripherals.....</b>	<b>36</b>
<b>3.2. ADC .....</b>	<b>36</b>
3.2.1. Descriptions of Peripheral registers.....	36
3.2.2. Descriptions of Peripheral functions .....	37
<b>3.3. CAN .....</b>	<b>36</b>
3.3.1. Descriptions of Peripheral registers.....	79
3.3.2. Descriptions of Peripheral functions .....	80
<b>3.4. CRC .....</b>	<b>36</b>
3.4.1. Descriptions of Peripheral registers.....	98
3.4.2. Descriptions of Peripheral functions .....	99
<b>3.5. CTC.....</b>	<b>103</b>
3.5.1. Descriptions of Peripheral registers.....	103
3.5.2. Descriptions of Peripheral functions .....	103
<b>3.6. DAC .....</b>	<b>119</b>
3.6.1. Descriptions of Peripheral registers.....	119
3.6.2. Descriptions of Peripheral functions .....	120
<b>3.7. DBG .....</b>	<b>138</b>
3.7.1. Descriptions of Peripheral registers.....	138
3.7.2. Descriptions of Peripheral functions .....	138

<b>3.8. DCI</b>	<b>144</b>
3.8.1. Descriptions of Peripheral registers	144
3.8.2. Descriptions of Peripheral functions	145
<b>3.9. DMA</b>	<b>161</b>
3.9.1. Descriptions of Peripheral registers	161
3.9.2. Descriptions of Peripheral functions	162
<b>3.10. ENET</b>	<b>192</b>
3.10.1. Descriptions of Peripheral registers	192
3.10.2. Descriptions of Peripheral functions	195
<b>3.11. EXMC</b>	<b>286</b>
3.11.1. Descriptions of Peripheral registers	287
3.11.2. Descriptions of Peripheral functions	287
<b>3.12. EXTI</b>	<b>322</b>
3.12.1. Descriptions of Peripheral registers	323
3.12.2. Descriptions of Peripheral functions	323
<b>3.13. FMC</b>	<b>330</b>
3.13.1. Descriptions of Peripheral registers	330
3.13.2. Descriptions of Peripheral functions	331
<b>3.14. FWDGT</b>	<b>355</b>
3.14.1. Descriptions of Peripheral registers	355
3.14.2. Descriptions of Peripheral functions	355
<b>3.15. GPIO</b>	<b>361</b>
3.15.1. Descriptions of Peripheral registers	361
3.15.2. Descriptions of Peripheral functions	361
<b>3.16. I2C</b>	<b>372</b>
3.16.1. Descriptions of Peripheral registers	372
3.16.2. Descriptions of Peripheral functions	373
<b>3.17. IPA</b>	<b>405</b>
3.17.1. Descriptions of Peripheral registers	405
3.17.2. Descriptions of Peripheral functions	406
<b>3.18. IREF</b>	<b>429</b>
3.18.1. Descriptions of Peripheral registers	429
3.18.2. Descriptions of Peripheral functions	429
<b>3.19. MISC</b>	<b>433</b>
3.19.1. Descriptions of Peripheral registers	434
3.19.2. Descriptions of Peripheral functions	435
<b>3.20. PMU</b>	<b>441</b>
3.20.1. Descriptions of Peripheral registers	441
3.20.2. Descriptions of Peripheral functions	442

<b>3.21.</b>	<b>RCU .....</b>	<b>453</b>
3.21.1.	Descriptions of Peripheral registers .....	454
3.21.2.	Descriptions of Peripheral functions .....	455
<b>3.22.</b>	<b>RTC .....</b>	<b>495</b>
3.22.1.	Descriptions of Peripheral registers .....	495
3.22.2.	Descriptions of Peripheral functions .....	496
<b>3.23.</b>	<b>SDIO .....</b>	<b>530</b>
3.23.1.	Descriptions of Peripheral registers .....	530
3.23.2.	Descriptions of Peripheral functions .....	531
<b>3.24.</b>	<b>SPI .....</b>	<b>569</b>
3.24.1.	Descriptions of Peripheral registers .....	569
3.24.2.	Descriptions of Peripheral functions .....	570
<b>3.25.</b>	<b>SYSCFG .....</b>	<b>596</b>
3.25.1.	Descriptions of Peripheral registers .....	596
3.25.2.	Descriptions of Peripheral functions .....	597
<b>3.26.</b>	<b>TIMER.....</b>	<b>603</b>
3.26.1.	Descriptions of Peripheral registers .....	603
3.26.2.	Descriptions of Peripheral functions .....	604
<b>3.27.</b>	<b>TLI .....</b>	<b>676</b>
3.27.1.	Descriptions of Peripheral registers .....	676
3.27.2.	Descriptions of Peripheral functions .....	677
<b>3.28.</b>	<b>TRNG.....</b>	<b>699</b>
3.28.1.	Descriptions of Peripheral registers .....	700
3.28.2.	Descriptions of Peripheral functions .....	700
<b>3.29.</b>	<b>USART.....</b>	<b>706</b>
3.29.1.	Descriptions of Peripheral registers .....	706
3.29.2.	Descriptions of Peripheral functions .....	707
<b>3.30.</b>	<b>WWDGT.....</b>	<b>742</b>
3.30.1.	Descriptions of Peripheral registers .....	742
3.30.2.	Descriptions of Peripheral functions .....	743
<b>4.</b>	<b>Revision history .....</b>	<b>749</b>



## List of Figures

Figure 2-1. File structure of firmware library of GD32A490 .....	30
Figure 2-2. Select peripheral example files .....	32
Figure 2-3. Copy the peripheral example files .....	33
Figure 2-4. Open the project file .....	33
Figure 2-5. Configure project files .....	34
Figure 2-6. Compile-debug-download .....	34

# List of Tables

Table 1-1. Peripherals .....	27
Table 2-1. Function descriptions of Firmware Library .....	35
Table 3-1. Peripheral function format of Firmware Library .....	36
Table 3-2. ADC Registers .....	36
Table 3-3. ADC firmware function.....	37
Table 3-4. Function adc_deinit.....	38
Table 3-5. Function adc_clock_config.....	39
Table 3-6. Function adc_special_function_config.....	40
Table 3-7. Function adc_data_alignment_config.....	41
Table 3-8. Function adc_enable .....	42
Table 3-9. Function adc_disable .....	42
Table 3-10. Function adc_calibration_enable.....	43
Table 3-11. Function adc_channel_16_to_18 .....	44
Table 3-12. Function adc_resolution_config .....	45
Table 3-13. Function adc_oversample_mode_config .....	46
Table 3-14. Function adc_oversample_mode_enable .....	48
Table 3-15. Function adc_oversample_mode_disable .....	48
Table 3-16. Function adc_dma_mode_enable.....	49
Table 3-17. Function adc_dma_mode_disable.....	50
Table 3-18. Function adc_dma_request_after_last_enable .....	50
Table 3-19. Function adc_dma_mode_disable.....	51
Table 3-20. Function adc_discontinuous_mode_config .....	52
Table 3-21. Function adc_channel_length_config.....	53
Table 3-22. Function adc_regular_channel_config .....	53
Table 3-23. Function adc_inserted_channel_config .....	55
Table 3-24. Function adc_inserted_channel_offset_config.....	56
Table 3-25. Function adc_external_trigger_source_config .....	57
Table 3-26. Function adc_external_trigger_config.....	60
Table 3-27. Function adc_software_trigger_enable .....	61
Table 3-28. Function adc_end_of_conversion_config.....	62
Table 3-29. Function adc_regular_data_read.....	63
Table 3-30. Function adc_inserted_data_read .....	63
Table 3-31. Function adc_watchdog_single_channel_disable.....	64
Table 3-32. Function adc_watchdog_single_channel_enable.....	65
Table 3-33. Function adc_watchdog_group_channel_enable.....	66
Table 3-34. Function adc_watchdog_disable .....	67
Table 3-35. Function adc_watchdog_threshold_config .....	67
Table 3-36. Function adc_flag_get .....	68
Table 3-37. Function adc_flag_clear .....	69
Table 3-38. Function adc_regular_software_startconv_flag_get.....	70
Table 3-39. Function adc_inserted_software_startconv_flag_get.....	70

Table 3-40. Function adc_interrupt_flag_get.....	71
Table 3-41. Function adc_interrupt_flag_clear .....	72
Table 3-42. Function adc_interrupt_enable .....	73
Table 3-43. Function adc_interrupt_disable .....	74
Table 3-44. Function adc_sync_mode_config.....	74
Table 3-45. Function adc_interrupt_disable .....	76
Table 3-46. Function adc_sync_dma_config .....	77
Table 3-47. Function adc_sync_dma_request_after_last_enable .....	77
Table 3-48. Function adc_sync_dma_request_after_last_disable .....	78
Table 3-49. Function adc_interrupt_disable .....	79
Table 3-50. CAN Registers .....	79
Table 3-51. CAN firmware function .....	80
Table 3-52. can_parameter_struct .....	81
Table 3-53. can_transmit_message_struct .....	81
Table 3-54. can_receive_message_struct .....	82
Table 3-55. can_filter_parameter_struct .....	82
Table 3-56. Function can_deinit.....	82
Table 3-57. Function can_struct_para_init .....	83
Table 3-58. Function can_init .....	83
Table 3-59. Function can_filter_init .....	84
Table 3-60. Function can1_filter_start_bank .....	84
Table 3-61. Function can_debug_freeze_enable .....	85
Table 3-62. Function can_debug_freeze_disable .....	85
Table 3-63. Function can_time_trigger_mode_enable.....	86
Table 3-64. Function can_time_trigger_mode_disable.....	86
Table 3-65. Function can_message_transmit.....	87
Table 3-66. Function can_transmit_states .....	88
Table 3-67. Function can_transmission_stop .....	88
Table 3-68. Function can_message_receive .....	89
Table 3-69. Function can_fifo_release .....	89
Table 3-70. Function can_receive_message_length_get .....	90
Table 3-71. Function can_working_mode_set.....	91
Table 3-72. Function can_wakeup .....	91
Table 3-73. Function can_error_get .....	92
Table 3-74. Function can_receive_error_number_get .....	92
Table 3-75. Function can_transmit_error_number_get .....	93
Table 3-76. Function can_flag_get .....	93
Table 3-77. Function can_flag_clear .....	94
Table 3-78. Function can_interrupt_enable .....	95
Table 3-79. Function can_interrupt_disable .....	96
Table 3-80. Function can_interrupt_flag_get.....	97
Table 3-81. Function can_interrupt_flag_clear .....	97
Table 3-82. CRC Registers .....	99
Table 3-83. CRC firmware function .....	99

Table 3-84. Function <code>crc_deinit</code> .....	99
Table 3-85. Function <code>crc_data_register_reset</code> .....	100
Table 3-86. Function <code>crc_data_register_read</code> .....	100
Table 3-87. Function <code>crc_free_data_register_read</code> .....	101
Table 3-88. Function <code>crc_free_data_register_write</code> .....	101
Table 3-89. Function <code>crc_single_data_calculate</code> .....	102
Table 3-90. Function <code>crc_block_data_calculate</code> .....	102
Table 3-91. CTC Registers .....	103
Table 3-92. CTC firmware function.....	103
Table 3-93. Function <code>ctc_deinit</code> .....	104
Table 3-94. Function <code>ctc_counter_enable</code> .....	105
Table 3-95. Function <code>ctc_counter_disable</code> .....	105
Table 3-96. Function <code>ctc_irc48m_trim_value_config</code> .....	106
Table 3-97. Function <code>ctc_software_refsource_pulse_generate</code> .....	107
Table 3-98. Function <code>ctc_hardware_trim_mode_config</code> .....	107
Table 3-99. Function <code>ctc_refsource_polarity_config</code> .....	108
Table 3-101. Function <code>ctc_refsource_signal_select</code> .....	109
Table 3-102. Function <code>ctc_refsource_prescaler_config</code> .....	109
Table 3-103. Function <code>ctc_clock_limit_value_config</code> .....	110
Table 3-104. Function <code>ctc_counter_reload_value_config</code> .....	111
Table 3-105. Function <code>ctc_counter_capture_value_read</code> .....	112
Table 3-106. Function <code>ctc_counter_direction_read</code> .....	112
Table 3-107. Function <code>ctc_counter_reload_value_read</code> .....	113
Table 3-108. Function <code>ctc_irc48m_trim_value_read</code> .....	114
Table 3-109. Function <code>ctc_interrupt_enable</code> .....	114
Table 3-110. Function <code>ctc_interrupt_disable</code> .....	115
Table 3-111. Function <code>ctc_interrupt_flag_get</code> .....	116
Table 3-112. Function <code>ctc_interrupt_flag_clear</code> .....	117
Table 3-113. Function <code>ctc_flag_get</code> .....	118
Table 3-114. Function <code>ctc_flag_clear</code> .....	118
Table 3-115. DAC Registers .....	119
Table 3-116. DAC firmware function.....	错误!未定义书签。
Table 3-117. Function <code>dac_deinit</code> .....	错误!未定义书签。
Table 3-118. Function <code>dac_enable</code> .....	错误!未定义书签。
Table 3-119. Function <code>dac_disable</code> .....	错误!未定义书签。
Table 3-120. Function <code>dac_dma_enable</code> .....	错误!未定义书签。
Table 3-121. Function <code>dac_dma_disable</code> .....	错误!未定义书签。
Table 3-122. Function <code>dac_output_buffer_enable</code> .....	错误!未定义书签。
Table 3-123. Function <code>dac_output_buffer_disable</code> .....	错误!未定义书签。
Table 3-124. Function <code>dac_output_value_get</code> .....	错误!未定义书签。
Table 3-125. Function <code>dac_data_set</code> .....	错误!未定义书签。
Table 3-126. Function <code>dac_trigger_enable</code> .....	错误!未定义书签。
Table 3-127. Function <code>dac_trigger_disable</code> .....	错误!未定义书签。
Table 3-128. Function <code>dac_trigger_source_config</code> .....	错误!未定义书签。

Table 3-129. Function <code>dac_software_trigger_enable</code> .....	错误!未定义书签。
Table 3-130. Function <code>dac_software_trigger_disable</code> .....	错误!未定义书签。
Table 3-131. Function <code>dac_wave_mode_config</code> .....	错误!未定义书签。
Table 3-132. Function <code>dac_wave_bit_width_config</code> .....	错误!未定义书签。
Table 3-133. Function <code>dac_ifsr_noise_config</code> .....	错误!未定义书签。
Table 3-134. Function <code>dac_triangle_noise_config</code> .....	错误!未定义书签。
Table 3-135. Function <code>dac_concurrent_enable</code> .....	错误!未定义书签。
Table 3-136. Function <code>dac_concurrent_disable</code> .....	错误!未定义书签。
Table 3-137. Function <code>dac_concurrent_software_trigger_enable</code> .....	错误!未定义书签。
Table 3-138. Function <code>dac_concurrent_software_trigger_disable</code> .....	错误!未定义书签。
Table 3-139. Function <code>dac_concurrent_output_buffer_enable</code> .....	错误!未定义书签。
Table 3-140. Function <code>dac_concurrent_output_buffer_disable</code> .....	错误!未定义书签。
Table 3-141. Function <code>dac_concurrent_data_set</code> .....	错误!未定义书签。
Table 3-142. Function <code>dac_concurrent_interrupt_enable</code> .....	错误!未定义书签。
Table 3-143. Function <code>dac_concurrent_interrupt_disable</code> .....	错误!未定义书签。
Table 3-144. Function <code>dac_flag_get</code> .....	错误!未定义书签。
Table 3-145. Function <code>dac_flag_clear</code> .....	错误!未定义书签。
Table 3-146. Function <code>dac_interrupt_enable</code> .....	错误!未定义书签。
Table 3-147. Function <code>dac_interrupt_disable</code> .....	错误!未定义书签。
Table 3-148. Function <code>dac_interrupt_flag_get</code> .....	错误!未定义书签。
Table 3-149. Function <code>dac_interrupt_flag_get</code> .....	错误!未定义书签。
Table 3-150. DBG Registers .....	138
Table 3-151. DBG firmware function .....	138
Table 3-152. Enum <code>dbg_periph_enum</code> .....	138
Table 3-153. Function <code>dbg_id_get</code> .....	139
Table 3-154. Function <code>dbg_low_power_enable</code> .....	140
Table 3-155. Function <code>dbg_low_power_disable</code> .....	141
Table 3-156. Function <code>dbg_periph_enable</code> .....	141
Table 3-157. Function <code>dbg_periph_disable</code> .....	142
Table 3-158. Function <code>dbg_trace_pin_enable</code> .....	143
Table 3-159. Function <code>dbg_trace_pin_disable</code> .....	144
Table 3-160. Function <code>dbg_trace_pin_mode_set</code> .....	错误!未定义书签。
Table 3-161. DCI Registers .....	144
Table 3-162. DCI firmware function .....	145
Table 3-163. Structure <code>dc_i_parameter_struct</code> .....	146
Table 3-164. Function <code>dc_i_deinit</code> .....	146
Table 3-165. Function <code>dc_i_init</code> .....	147
Table 3-166. Function <code>dc_i_enable</code> .....	148
Table 3-167. Function <code>dc_i_disable</code> .....	148
Table 3-168. Function <code>dc_i_capture_enable</code> .....	149
Table 3-169. Function <code>dc_i_capture_disable</code> .....	150
Table 3-170. Function <code>dc_i_jpeg_enable</code> .....	150
Table 3-171. Function <code>dc_i_jpeg_disable</code> .....	151
Table 3-172. Function <code>dc_i_crop_window_enable</code> .....	151

Table 3-173. Function dci_crop_window_disable .....	152
Table 3-174. Function dci_crop_window_config .....	153
Table 3-175. Function dci_embedded_sync_enable .....	153
Table 3-176. Function dci_embedded_sync_disable .....	154
Table 3-177. Function dci_sync_codes_config .....	155
Table 3-178. Function dci_sync_codes_unmask_config .....	155
Table 3-179. Function dci_data_read .....	156
Table 3-180. Function dci_flag_get .....	157
Table 3-181. Function dci_interrupt_enable .....	158
Table 3-182. Function dci_interrupt_disable .....	159
Table 3-183. Function dci_interrupt_flag_get .....	159
Table 3-184. Function dci_interrupt_flag_clear .....	160
Table 3-185. DMA Registers .....	161
Table 3-186. DMA firmware function .....	162
Table 3-187. Structure dma_multi_data_parameter_struct .....	163
Table 3-188. Structure dma_single_data_parameter_struct .....	164
Table 3-189. Function dma_deinit .....	164
Table 3-190. Function dma_single_data_mode_init .....	165
Table 3-191. Function dma_multi_data_mode_init .....	166
Table 3-192. Function dma_periph_address_config .....	167
Table 3-193. Function dma_memory_address_config .....	167
Table 3-194. Function dma_transfer_number_config .....	168
Table 3-195. Function dma_transfer_number_get .....	169
Table 3-196. Function dma_priority_config .....	170
Table 3-197. Function dma_memory_burst_beats_config .....	171
Table 3-198. Function dma_periph_burst_beats_config .....	172
Table 3-199. Function dma_memory_width_config .....	173
Table 3-200. Function dma_periph_width_config .....	174
Table 3-201. Function dma_memory_address_generation_config .....	175
Table 3-202. Function dma_peripheral_address_generation_config .....	176
Table 3-203. Function dma_circulation_enable .....	177
Table 3-204. Function dma_circulation_disable .....	178
Table 3-205. Function dma_channel_enable .....	178
Table 3-206. Function dma_channel_disable .....	179
Table 3-207. Function dma_transfer_direction_config .....	180
Table 3-208. Function dma_switch_buffer_mode_config .....	181
Table 3-209. Function dma_using_memory_get .....	182
Table 3-210. Function dma_channel_subperipheral_select .....	182
Table 3-211. Function dma_flow_controller_config .....	183
Table 3-212. Function dma_switch_buffer_mode_enable .....	184
Table 3-213. Function dma_fifo_status_get .....	185
Table 3-214. Function dma_flag_get .....	186
Table 3-215. Function dma_flag_clear .....	187
Table 3-216. Function dma_interrupt_flag_get .....	188

Table 3-217. Function dma_interrupt_flag_clear .....	189
Table 3-218. Function dma_interrupt_enable.....	190
Table 3-219. Function dma_interrupt_disable.....	191
Table 3-220. ENET Registers .....	192
Table 3-221. ENET firmware function .....	195
Table 3-222. Structure enet_initpara_struct .....	199
Table 3-223. Structure enet_descriptors_struct .....	200
Table 3-224. Structure enet_ptp_systime_struct.....	200
Table 3-225. Function enet_deinit .....	200
Table 3-226. Function enet_initpara_config .....	201
Table 3-227. Function enet_init.....	205
Table 3-228. Function enet_software_reset.....	207
Table 3-229. Function enet_rxframe_size_get.....	208
Table 3-230. Function enet_descriptors_chain_init.....	208
Table 3-231. Function enet_descriptors_ring_init.....	209
Table 3-232. Function enet_frame_receive .....	210
Table 3-233. Function enet_frame_transmit .....	210
Table 3-234. Function enet_transmit_checksum_config .....	211
Table 3-235. Function enet_enable .....	212
Table 3-236. Function enet_disable .....	213
Table 3-237. Function enet_mac_address_set.....	213
Table 3-238. Function enet_mac_address_get .....	214
Table 3-239. Function enet_flag_get.....	215
Table 3-240. Function enet_flag_clear.....	218
Table 3-241. Function enet_interrupt_enable.....	219
Table 3-242. Function enet_interrupt_disable.....	221
Table 3-243. Function enet_interrupt_flag_get .....	223
Table 3-244. Function enet_interrupt_flag_clear .....	225
Table 3-245. Function enet_tx_enable .....	227
Table 3-246. Function enet_tx_disable .....	227
Table 3-247. Function enet_rx_enable .....	228
Table 3-248. Function enet_rx_disable.....	229
Table 3-249. Function enet_registers_get.....	229
Table 3-250. Function enet_debug_status_get .....	230
Table 3-251. Function enet_address_filter_enable.....	232
Table 3-252. Function enet_address_filter_disable .....	232
Table 3-253. Function enet_address_filter_config .....	233
Table 3-254. Function enet_phy_config .....	235
Table 3-255. Function enet_phy_write_read.....	235
Table 3-256. Function enet_phyloopback_enable .....	236
Table 3-257. Function enet_phyloopback_disable .....	237
Table 3-258. Function enet_forward_feature_enable.....	237
Table 3-259. Function enet_forward_feature_disable.....	238
Table 3-260. Function enet_fliter_feature_enable .....	239



Table 3-261. Function enet_fliter_feature_disable .....	240
Table 3-262. Function enet_pauseframe_generate .....	241
Table 3-263. Function enet_pauseframe_detect_config .....	242
Table 3-264. Function enet_pauseframe_config .....	243
Table 3-265. Function enet_flowcontrol_threshold_config .....	244
Table 3-266. Function enet_flowcontrol_feature_enable .....	245
Table 3-267. Function enet_flowcontrol_feature_disable .....	246
Table 3-268. Function enet_dmaprocess_state_get .....	247
Table 3-269. Function enet_dmaprocess_resume .....	248
Table 3-270. Function enet_rxprocess_check_recovery .....	249
Table 3-271. Function enet_txfifo_flush .....	249
Table 3-272. Function enet_current_desc_address_get .....	250
Table 3-273. Function enet_desc_information_get .....	251
Table 3-274. Function enet_missed_frame_counter_get .....	252
Table 3-275. Function enet_desc_flag_get .....	253
Table 3-276. Function enet_desc_flag_set .....	255
Table 3-277. Function enet_desc_flag_clear .....	256
Table 3-278. Function enet_rx_desc_immediate_receive_complete_interrupt .....	258
Table 3-279. Function enet_rx_desc_delay_receive_complete_interrupt .....	258
Table 3-280. Function enet_rxframe_drop .....	259
Table 3-281. Function enet_dma_feature_enable .....	260
Table 3-282. Function enet_dma_feature_disable .....	261
Table 3-283. Function enet_rx_desc_enhanced_status_get .....	261
Table 3-284. Function enet_desc_select_enhanced_mode .....	262
Table 3-285. Function enet_ptp_enhanced_descriptors_chain_init .....	263
Table 3-286. Function enet_ptp_enhanced_descriptors_ring_init .....	264
Table 3-287. Function enet_ptpframe_receive_enhanced_mode .....	265
Table 3-288. Function enet_ptpframe_transmit_enhanced_mode .....	265
Table 3-289. Function enet_desc_select_normal_mode .....	266
Table 3-290. Function enet_ptp_normal_descriptors_chain_init .....	267
Table 3-291. Function enet_ptp_normal_descriptors_ring_init .....	268
Table 3-292. Function enet_ptpframe_receive_normal_mode .....	269
Table 3-293. Function enet_ptpframe_transmit_normal_mode .....	269
Table 3-294. Function enet_wum_filter_register_pointer_reset .....	270
Table 3-295. Function enet_wum_filter_config .....	271
Table 3-296. Function enet_wum_feature_enable .....	272
Table 3-297. Function enet_wum_feature_disable .....	272
Table 3-298. Function enet_msc_counters_reset .....	273
Table 3-299. Function enet_msc_feature_enable .....	274
Table 3-300. Function enet_msc_feature_disable .....	275
Table 3-301. Function enet_msc_counters_preset_config .....	275
Table 3-302. Function enet_msc_counters_get .....	276
Table 3-303. Function enet_ptp_feature_enable .....	277
Table 3-304. Function enet_ptp_feature_disable .....	278



Table 3-305. Function enet_ptp_timestamp_function_config .....	279
Table 3-306. Function enet_ptp_subsecond_increment_config .....	281
Table 3-307. Function enet_ptp_timestamp_addend_config .....	282
Table 3-308. Function enet_ptp_timestamp_update_config .....	282
Table 3-309. Function enet_ptp_expected_time_config .....	283
Table 3-310. Function enet_ptp_system_time_get .....	284
Table 3-311. Function enet_ptp_pps_output_frequency_config .....	285
Table 3-312. Function enet_initpara_reset .....	286
Table 3-313. EXMC Registers .....	287
Table 3-314. EXMC firmware function .....	287
Table 3-315. Structure exmc_norsram_timing_parameter_struct .....	289
Table 3-316. Structure exmc_norsram_parameter_struct .....	289
Table 3-317. Structure exmc_nand_pccard_timing_parameter_struct .....	290
Table 3-318. Structure exmc_nand_parameter_struct .....	290
Table 3-319. Structure exmc_pccard_parameter_struct .....	290
Table 3-320. Structure exmc_sdram_timing_parameter_struct .....	290
Table 3-321. Structure exmc_sdram_parameter_struct .....	291
Table 3-322. Structure exmc_sdram_command_parameter_struct .....	291
Table 3-323. Structure exmc_sqpsram_parameter_struct .....	292
Table 3-324. Function exmc_norsram_deinit .....	292
Table 3-325. Function exmc_norsram_struct_para_init .....	292
Table 3-326. Function exmc_norsram_init .....	293
Table 3-327. Function exmc_norsram_enable .....	294
Table 3-328. Function exmc_norsram_disable .....	295
Table 3-329. Function exmc_nand_deinit .....	295
Table 3-330. Function exmc_nand_struct_para_init .....	296
Table 3-331. Function exmc_nand_init .....	296
Table 3-332. Function exmc_nand_enable .....	297
Table 3-333. Function exmc_nand_disable .....	298
Table 3-334. Function exmc_pccard_deinit .....	298
Table 3-335. Function exmc_pccard_struct_para_init .....	299
Table 3-336. Function exmc_pccard_init .....	299
Table 3-337. Function exmc_pccard_enable .....	300
Table 3-338. Function exmc_pccard_disable .....	301
Table 3-339. Function exmc_sdram_deinit .....	301
Table 3-340. Function exmc_sdram_struct_para_init .....	302
Table 3-341. Function exmc_sdram_init .....	302
Table 3-369. Function exmc_sdram_struct_command_para_init .....	304
Table 3-342. Function exmc_sqpsram_deinit .....	304
Table 3-343. Function exmc_sqpsram_struct_para_init .....	305
Table 3-344. Function exmc_sqpsram_init .....	305
Table 3-345. Function exmc_norsram_consecutive_clock_config .....	306
Table 3-346. Function exmc_norsram_page_size_config .....	307
Table 3-347. Function exmc_nand_ecc_config .....	307

Table 3-348. Function <code>exmc_ecc_get</code> .....	308
Table 3-349. Function <code>exmc_sdram_readsample_enable</code> .....	309
Table 3-350. Function <code>exmc_sdram_readsample_config</code> .....	309
Table 3-351. Function <code>exmc_sdram_command_config</code> .....	310
Table 3-352. Function <code>exmc_sdram_refresh_count_set</code> .....	311
Table 3-353. Function <code>exmc_sdram_autorefresh_number_set</code> .....	311
Table 3-354. Function <code>exmc_sdram_write_protection_config</code> .....	312
Table 3-355. Function <code>exmc_sdram_bankstatus_get</code> .....	312
Table 3-356. Function <code>exmc_sqpsram_read_command_set</code> .....	313
Table 3-357. Function <code>exmc_sqpsram_write_command_set</code> .....	313
Table 3-358. Function <code>exmc_sqpsram_read_id_command_send</code> .....	314
Table 3-359. Function <code>exmc_sqpsram_write_cmd_send</code> .....	315
Table 3-360. Function <code>exmc_sqpsram_low_id_get</code> .....	315
Table 3-361. Function <code>exmc_sqpsram_low_id_get</code> .....	316
Table 3-362. Function <code>exmc_sqpsram_send_command_state_get</code> .....	316
Table 3-363. Function <code>exmc_interrupt_enable</code> .....	317
Table 3-364. Function <code>exmc_interrupt_disable</code> .....	318
Table 3-365. Function <code>exmc_flag_get</code> .....	319
Table 3-366. Function <code>exmc_flag_clear</code> .....	320
Table 3-367. Function <code>exmc_interrupt_flag_get</code> .....	321
Table 3-368. Function <code>exmc_interrupt_flag_clear</code> .....	322
Table 3-369. EXTI Registers.....	323
Table 3-370. EXTI firmware function .....	323
Table 3-371. Function <code>exti_deinit</code> .....	324
Table 3-372. Function <code>exti_init</code> .....	325
Table 3-373. Function <code>exti_interrupt_enable</code> .....	325
Table 3-374. Function <code>exti_interrupt_disable</code> .....	326
Table 3-375. Function <code>exti_event_enable</code> .....	326
Table 3-376. Function <code>exti_event_disable</code> .....	327
Table 3-377. Function <code>exti_software_interrupt_enable</code> .....	327
Table 3-378. Function <code>exti_software_interrupt_disable</code> .....	328
Table 3-379. Function <code>exti_flag_get</code> .....	328
Table 3-380. Function <code>exti_flag_clear</code> .....	329
Table 3-381. Function <code>exti_interrupt_flag_get</code> .....	329
Table 3-382. Function <code>exti_interrupt_flag_clear</code> .....	330
Table 3-383. FMC Registers .....	330
Table 3-384. FMC firmware function .....	331
Table 3-385. <code>fmc_state_enum</code> .....	332
Table 3-386. Function <code>fmc_wsnt_set</code> .....	332
Table 3-387. Function <code>fmc_unlock</code> .....	333
Table 3-388. Function <code>fmc_lock</code> .....	333
Table 3-389. Function <code>fmc_page_erase</code> .....	334
Table 3-390. Function <code>fmc_sector_erase</code> .....	334
Table 3-391. Function <code>fmc_mass_erase</code> .....	335

Table 3-392. Function fmc_bank0_erase .....	335
Table 3-393. Function fmc_bank1_erase .....	336
Table 3-394. Function fmc_word_program .....	336
Table 3-395. Function fmc_halfword_program .....	337
Table 3-396. Function fmc_halfword_program .....	337
Table 3-397. Function ob_unlock.....	338
Table 3-398. Function ob_lock .....	338
Table 3-399. Function ob_start.....	339
Table 3-400. Function ob_erase .....	339
Table 3-401. Function ob_write_protection_enable .....	340
Table 3-402. Function ob_write_protection_disable .....	341
Table 3-403. Function ob_drp_enable.....	341
Table 3-404. Function ob_drp_disable.....	342
Table 3-405. Function ob_security_protection_config .....	343
Table 3-406. Function ob_user_write .....	343
Table 3-407. Function ob_user_bor_threshold .....	344
Table 3-408. Function ob_boot_mode_config .....	345
Table 3-409. Function ob_user_get .....	346
Table 3-410. Function ob_write_protection0_get .....	347
Table 3-411. Function ob_write_protection1_get.....	347
Table 3-412. Function ob_drp0_get .....	348
Table 3-413. Function ob_drp0_get .....	348
Table 3-414. Function ob_spc_get.....	349
Table 3-415. Function ob_user_bor_threshold_get .....	349
Table 3-416. Function fmc_flag_get .....	350
Table 3-417. Function fmc_flag_clear .....	350
Table 3-418. Function fmc_interrupt_enable .....	351
Table 3-419. Function fmc_interrupt_disable .....	352
Table 3-420. Function fmc_interrupt_flag_get.....	352
Table 3-421. Function fmc_interrupt_flag_clear .....	353
Table 3-422. Function fmc_state_get .....	354
Table 3-423. Function fmc_ready_wait .....	354
Table 3-425. FWDGT Registers .....	355
Table 3-426. FWDGT firmware function .....	355
Table 3-427. Function fwdgt_write_enable .....	356
Table 3-428. Function fwdgt_write_disable .....	356
Table 3-429. Function fwdgt_enable .....	357
Table 3-430. Function fwdgt_prescaler_value_config .....	357
Table 3-431. Function fwdgt_reload_value_config.....	358
Table 3-432. Function fwdgt_counter_reload .....	358
Table 3-433. Function fwdgt_config.....	359
Table 3-434. Function fwdgt_flag_get fwdgt_write_disable .....	360
Table 3-432. GPIO Registers.....	361
Table 3-433. GPIO firmware function .....	361

Table 3-434. Function gpio_deinit .....	362
Table 3-435. Function gpio_mode_set.....	362
Table 3-436. Function gpio_output_options_set .....	363
Table 3-437. Function gpio_bit_set .....	364
Table 3-438. Function gpio_bit_reset .....	365
Table 3-439. Function gpio_bit_write.....	366
Table 3-440. Function gpio_port_write .....	366
Table 3-441. Function gpio_input_bit_get.....	367
Table 3-442. Function gpio_input_port_get.....	367
Table 3-443. Function gpio_output_bit_get .....	368
Table 3-444. Function gpio_output_port_get .....	369
Table 3-445. Function gpio_pin_remap_config.....	369
Table 3-446. Function gpio_pin_lock.....	370
Table 3-447. Function gpio_bit_toggle .....	371
Table 3-448. Function gpio_port_toggle .....	371
Table 3-449. I2C Registers .....	372
Table 3-450. I2C firmware function.....	373
Table 3-451. Function i2c_deinit .....	374
Table 3-452. Function i2c_clock_config.....	375
Table 3-453. Function i2c_mode_addr_config .....	376
Table 3-454. Function i2c_smbus_type_config .....	377
Table 3-455. Function i2c_ack_config .....	378
Table 3-456. Function i2c_ackpos_config .....	378
Table 3-457. Function i2c_master_addressing .....	379
Table 3-458. Function i2c_dualaddr_enable .....	380
Table 3-459. Function i2c_enable .....	381
Table 3-460. Function i2c_disable .....	382
Table 3-461. Function i2c_start_on_bus .....	383
Table 3-462. Function i2c_stop_on_bus .....	383
Table 3-463. Function i2c_data_transmit .....	384
Table 3-464. Function i2c_data_receive .....	385
Table 3-465. Function i2c_dma_config.....	385
Table 3-466. Function i2c_dma_last_transfer_enable .....	386
Table 3-467. Function i2c_stretch_scl_low_config .....	387
Table 3-468. Function i2c_slave_response_to_gcall_config.....	388
Table 3-469. Function i2c_software_reset_config.....	388
Table 3-470. Function i2c_pec_config .....	389
Table 3-471. Function i2c_pec_transfer_config.....	390
Table 3-472. Function i2c_pec_value_get .....	391
Table 3-473. Function i2c_smbus_alert_config .....	392
Table 3-474. Function i2c_smbus_arb_config .....	392
Table 3-475. Function i2c_analog_noise_filter_disable .....	393
Table 3-476. Function i2c_analog_noise_filter_enable .....	394
Table 3-477. Function i2c_digital_noise_filter_config .....	394

Table 3-478. Function i2c_sam_enable .....	395
Table 3-479. Function i2c_sam_disable .....	396
Table 3-480. Function i2c_sam_timeout_enable.....	396
Table 3-481. Function i2c_sam_timeout_disable .....	397
Table 3-482. Function i2c_flag_get .....	398
Table 3-483. Function i2c_flag_clear .....	399
Table 3-484. Function i2c_interrupt_enable .....	401
Table 3-485. Function i2c_interrupt_disable .....	402
Table 3-486. Function i2c_interrupt_flag_get.....	402
Table 3-487. Function i2c_interrupt_flag_clear.....	404
Table 3-488. IPA Registers .....	406
Table 3-489. IPA firmware function.....	406
Table3-490. ipa_foreground_parameter_struct.....	408
Table3-491. ipa_background_parameter_struct .....	408
Table3-492. ipa_destination_parameter_struct.....	409
Table 3-493. Function ipa_deinit.....	409
Table 3-494. Function ipa_transfer_enable.....	410
Table 3-495. Function ipa_transfer_hangup_enable.....	411
Table 3-496. Function ipa_transfer_hangup_disable.....	411
Table 3-497. Function ipa_transfer_stop_enable.....	412
Table 3-498. Function ipa_transfer_stop_disable .....	412
Table 3-499. Function ipa_foreground_lut_loading_enable .....	413
Table 3-500. Function ipa_background_lut_loading_enable.....	414
Table 3-501. Function ipa_pixel_format_convert_mode_set.....	414
Table 3-502. Function ipa_foreground_struct_para_init .....	415
Table 3-503. Function ipa_foreground_init.....	416
Table 3-504. Function ipa_background_struct_para_init.....	417
Table 3-505. Function ipa_background_init .....	417
Table 3-506. Function ipa_destination_struct_para_init .....	418
Table 3-507. Function ipa_destination_init.....	419
Table 3-508. Function ipa_foreground_lut_init .....	420
Table 3-509. Function ipa_background_lut_init.....	421
Table 3-510. Function ipa_line_mark_config.....	422
Table 3-511. Function ipa_inter_timer_config .....	423
Table 3-512. Function ipa_interval_clock_num_config .....	423
Table 3-513. Function ipa_flag_get .....	424
Table 3-514. Function ipa_flag_clear .....	425
Table 3-515. Function ipa_interrupt_enable .....	426
Table 3-516. Function ipa_interrupt_disable .....	426
Table 3-517. Function ipa_interrupt_flag_get.....	427
Table 3-518. Function ipa_interrupt_flag_clear .....	428
Table 3-519. IREF Registers .....	429
Table 3-520. IREF firmware function .....	429
Table 3-521. Function iref_deinit .....	430

Table 3-522. Function <code>iref_enable</code> .....	430
Table 3-523. Function <code>iref_disable</code> .....	431
Table 3-524. Function <code>iref_mode_set</code> .....	431
Table 3-525. Function <code>iref_sink_set</code> .....	432
Table 3-526. Function <code>iref_precision_trim_value_set</code> .....	432
Table 3-527. Function <code>iref_step_data_config</code> .....	433
Table 3-528. NVIC Registers .....	434
Table 3-529. SysTick Registers .....	434
Table 3-530. Enum <code>IRQn_Type</code> .....	435
Table 3-531. MISC firmware function .....	437
Table 3-532. Function <code>nvic_priority_group_set</code> .....	437
Table 3-533. Function <code>nvic_irq_enable</code> .....	438
Table 3-534. Function <code>nvic_irq_disable</code> .....	438
Table 3-535. Function <code>nvic_vector_table_set</code> .....	439
Table 3-536. Function <code>system_lowpower_set</code> .....	439
Table 3-537. Function <code>system_lowpower_reset</code> .....	440
Table 3-538. Function <code>systick_clksource_set</code> .....	441
Table 3-539. PMU Registers .....	441
Table 3-540. PMU firmware function .....	442
Table 3-541. Function <code>pmu_deinit</code> .....	442
Table 3-542. Function <code>pmu_lvd_select</code> .....	443
Table 3-543. Function <code>pmu_ldo_output_select</code> .....	错误!未定义书签。
Table 3-544. Function <code>pmu_low_driver_mode_enable</code> .....	444
Table 3-545. Function <code>pmu_highdriver_switch_select</code> .....	445
Table 3-546. Function <code>pmu_highdriver_mode_enable</code> .....	444
Table 3-547. Function <code>pmu_highdriver_mode_disable</code> .....	445
Table 3-548. Function <code>pmu_lvd_disable</code> .....	错误!未定义书签。
Table 3-549. Function <code>pmu_lowdriver_lowpower_config</code> .....	447
Table 3-550. <code>pmu_lowdriver_normalpower_config</code> .....	447
Table 3-551. Function <code>pmu_to_sleepmode</code> .....	448
Table 3-552. Function <code>pmu_to_deepsleepmode</code> .....	449
Table 3-553. Function <code>pmu_to_standbymode</code> .....	449
Table 3-554. Function <code>pmu_backup_ldo_config</code> .....	451
Table 3-555. Function <code>pmu_flag_reset</code> .....	453
Table 3-556. Function <code>pmu_flag_get</code> .....	452
Table 3-557. Function <code>pmu_backup_write_enable</code> .....	451
Table 3-558. Function <code>pmu_backup_write_disable</code> .....	452
Table 3-559. Function <code>pmu_wakeup_pin_enable</code> .....	450
Table 3-560. Function <code>pmu_wakeup_pin_disable</code> .....	450
Table 3-561. RCU Registers .....	454
Table 3-562. RCU firmware function .....	455
Table 3-563. Enum <code>rcu_periph_enum</code> .....	457
Table 3-565. Enum <code>rcu_periph_sleep_enum</code> .....	错误!未定义书签。
Table 3-566. Enum <code>rcu_periph_reset_enum</code> .....	错误!未定义书签。

Table 3-567. Enum rcu_flag_enum .....	错误!未定义书签。
Table 3-568. Enum rcu_int_flag_enum .....	错误!未定义书签。
Table 3-569. Enum rcu_int_flag_clear_enum .....	错误!未定义书签。
Table 3-570. Enum rcu_int_enum .....	错误!未定义书签。
Table 3-571. Enum rcu_osci_type_enum .....	错误!未定义书签。
Table 3-572. Enum rcu_clock_freq_enum .....	错误!未定义书签。
Table 3-573. Function rcu_deinit .....	错误!未定义书签。
Table 3-574. Function rcu_periph_clock_enable .....	错误!未定义书签。
Table 3-575. Function rcu_periph_clock_disable .....	错误!未定义书签。
Table 3-576. Function rcu_periph_clock_sleep_enable .....	错误!未定义书签。
Table 3-577. Function rcu_periph_clock_sleep_disable .....	错误!未定义书签。
Table 3-578. Function rcu_periph_reset_enable .....	错误!未定义书签。
Table 3-579. Function rcu_periph_reset_disable .....	错误!未定义书签。
Table 3-580. Function rcu_bkp_reset_enable .....	错误!未定义书签。
Table 3-581. Function rcu_bkp_reset_disable .....	错误!未定义书签。
Table 3-582. Function rcu_system_clock_source_config .....	错误!未定义书签。
Table 3-583. Function rcu_system_clock_source_get .....	错误!未定义书签。
Table 3-584. Function rcu_ahb_clock_config .....	错误!未定义书签。
Table 3-585. Function rcu_apb1_clock_config .....	错误!未定义书签。
Table 3-586. Function rcu_apb2_clock_config .....	错误!未定义书签。
Table 3-587. Function rcu_ckout0_config .....	错误!未定义书签。
Table 3-588. Function rcu_ckout1_config .....	错误!未定义书签。
Table 3-589. Function rcu_pll_config .....	错误!未定义书签。
Table 3-590. Function rcu_plli2s_config .....	错误!未定义书签。
Table 3-591. Function rcu_pllsai_config .....	错误!未定义书签。
Table 3-592. Function rcu_rtc_clock_config .....	错误!未定义书签。
Table 3-593. Function rcu_i2s_clock_config .....	错误!未定义书签。
Table 3-594. Function rcu_ck48m_clock_config .....	错误!未定义书签。
Table 3-595. Function rcu_pll48m_clock_config .....	479
Table 3-596. Function rcu_timer_clock_prescaler_config .....	480
Table 3-597. Function rcu_tli_clock_div_config .....	480
Table 3-598. Function rcu_flag_get .....	481
Table 3-599. Function rcu_all_reset_flag_clear .....	482
Table 3-600. Function rcu_interrupt_flag_get .....	482
Table 3-601. Function rcu_interrupt_flag_clear .....	483
Table 3-602. Function rcu_interrupt_enable .....	484
Table 3-603. Function rcu_interrupt_disable .....	484
Table 3-604. Function rcu_lxtal_drive_capability_config .....	485
Table 3-605. Function rcu_osci_stab_wait .....	486
Table 3-606. Function rcu_osci_on .....	486
Table 3-607. Function rcu_osci_off .....	487
Table 3-608. Function rcu_osci_bypass_mode_enable .....	487
Table 3-609. Function rcu_osci_bypass_mode_disable .....	488
Table 3-610. Function rcu_hxtal_clock_monitor_enable .....	489



Table 3-611. Function rcu_hxtal_clock_monitor_disable.....	489
Table 3-612. Function rcu_irc16m_adjust_value_set.....	490
Table 3-613. Function rcu_spread_spectrum_config .....	491
Table 3-614. Function rcu_spread_spectrum_enable.....	492
Table 3-615. Function rcu_spread_spectrum_disable.....	492
Table 3-616. Function rcu_voltage_key_unlock .....	493
Table 3-617. Function rcu_deepsleep_voltage_set.....	493
Table 3-618. Function rcu_clock_freq_get.....	494
Table 3-619. RTC Registers .....	495
Table 3-620. RTC firmware function .....	496
Table 3-621. rtc_parameter_struct.....	498
Table 3-622. rtc_alarm_struct.....	498
Table 3-623. rtc_timestamp_struct.....	499
Table 3-624. rtc_tamper_struct .....	499
Table 3-625. Function rtc_deinit .....	499
Table 3-626. Function rtc_init.....	500
Table 3-627. Function rtc_init_mode_enter .....	501
Table 3-628. Function rtc_init_mode_exit.....	501
Table 3-629. Function rtc_register_sync_wait .....	502
Table 3-630. Function rtc_current_time_get.....	503
Table 3-631. Function rtc_subsecond_get.....	503
Table 3-632. Function rtc_alarm_config.....	504
Table 3-633. Function rtc_alarm_subsecond_config.....	505
Table 3-634. Function rtc_alarm_get.....	506
Table 3-635. Function rtc_alarm_subsecond_get .....	507
Table 3-636. Function rtc_alarm_enable .....	508
Table 3-637. Function rtc_alarm_disable .....	508
Table 3-638. Function rtc_timestamp_enable .....	509
Table 3-639. Function rtc_timestamp_disable .....	510
Table 3-640. Function rtc_timestamp_get.....	510
Table 3-641. Function rtc_timestamp_subsecond_get.....	511
Table 3-642. Function rtc_timestamp_pin_map.....	512
Table 3-643. Function rtc_timestamp_enable .....	512
Table 3-644. Function rtc_tamper_disable.....	513
Table 3-645. Function rtc_tamper0_pin_map .....	514
Table 3-646. Function rtc_interrupt_enable .....	514
Table 3-647. Function rtc_interrupt_disable.....	515
Table 3-648. Function rtc_flag_get.....	516
Table 3-649. Function rtc_flag_clear .....	517
Table 3-650. Function rtc_alter_output_config .....	518
Table 3-651. rtc_calibration_output_config .....	519
Table 3-652. rtc_hour_adjust.....	520
Table 3-653. rtc_second_adjust .....	520
Table 3-654. rtc_bypass_shadow_enable .....	521



Table 3-655. rtc_bypass_shadow_disable .....	522
Table 3-656. rtc_refclock_detection_enable .....	522
Table 3-657. rtc_refclock_detection_disable .....	523
Table 3-658. rtc_wakeup_enable .....	524
Table 3-659. rtc_wakeup_disable .....	524
Table 3-660. rtc_wakeup_clock_set .....	525
Table 3-661. rtc_wakeup_timer_set.....	526
Table 3-662. rtc_wakeup_timer_get .....	526
Table 3-663. rtc_smooth_calibration_config .....	527
Table 3-664. rtc_coarse_calibration_enable .....	528
Table 3-665. rtc_coarse_calibration_disable .....	528
Table 3-666. rtc_coarse_calibration_config.....	529
Table 3-667. SDIO Registers .....	530
Table 3-668. SDIO firmware function .....	531
Table 3-669. Function sdio_deinit .....	533
Table 3-670. Function sdio_clock_config .....	534
Table 3-671. Function sdio_hardware_clock_enable .....	535
Table 3-672. Function sdio_hardware_clock_disable.....	535
Table 3-673. Function sdio_bus_mode_set .....	536
Table 3-674. Function sdio_power_state_set.....	537
Table 3-675. Function sdio_power_state_get.....	537
Table 3-676. Function sdio_clock_enable.....	538
Table 3-677. Function sdio_clock_disable.....	539
Table 3-678. Function sdio_command_response_config .....	539
Table 3-679. Function sdio_wait_type_set.....	540
Table 3-680. Function sdio_csm_enable .....	541
Table 3-681. Function sdio_csm_disable.....	542
Table 3-682. Function sdio_command_index_get.....	542
Table 3-683. Function sdio_response_get .....	543
Table 3-684. Function sdio_data_config .....	544
Table 3-685. Function sdio_data_transfer_config .....	545
Table 3-686. Function sdio_dsm_enable.....	546
Table 3-687. Function sdio_dsm_disable.....	547
Table 3-688. Function sdio_data_write .....	547
Table 3-689. Function sdio_data_read.....	548
Table 3-690. Function sdio_data_counter_get .....	549
Table 3-691. Function sdio_data_counter_get .....	549
Table 3-692. Function sdio_dma_enable.....	550
Table 3-693. Function sdio_dma_disable.....	550
Table 3-694. Function sdio_flag_get .....	551
Table 3-695. Function sdio_flag_clear .....	553
Table 3-696. Function sdio_interrupt_enable .....	554
Table 3-697. Function sdio_interrupt_disable .....	555
Table 3-698. Function sdio_interrupt_flag_get .....	557

Table 3-699. Function <code>sdio_interrupt_flag_clear</code> .....	559
Table 3-700. Function <code>sdio_readwait_enable</code> .....	560
Table 3-701. Function <code>sdio_readwait_disable</code> .....	561
Table 3-702. Function <code>sdio_stop_readwait_enable</code> .....	561
Table 3-703. Function <code>sdio_stop_readwait_disable</code> .....	562
Table 3-704. Function <code>sdio_readwait_type_set</code> .....	562
Table 3-705. Function <code>sdio_operation_enable</code> .....	563
Table 3-706. Function <code>sdio_operation_disable</code> .....	564
Table 3-707. Function <code>sdio_suspend_enable</code> .....	564
Table 3-708. Function <code>sdio_suspend_disable</code> .....	565
Table 3-709. Function <code>sdio_ceata_command_enable</code> .....	566
Table 3-710. Function <code>sdio_ceata_command_disable</code> .....	566
Table 3-711. Function <code>sdio_ceata_interrupt_enable</code> .....	567
Table 3-712. Function <code>sdio_ceata_interrupt_disable</code> .....	567
Table 3-713. Function <code>sdio_ceata_command_completion_enable</code> .....	568
Table 3-714. Function <code>sdio_ceata_command_completion_disable</code> .....	569
Table 3-715. SPI/I2S Registers .....	569
Table 3-716. SPI/I2S firmware function .....	570
Table 3-717. <code>spi_parameter_struct</code> .....	571
Table 3-718. Function <code>spi_i2s_deinit</code> .....	572
Table 3-719. Function <code>spi_struct_para_init</code> .....	572
Table 3-720. Function <code>spi_init</code> .....	573
Table 3-721. Function <code>spi_enable</code> .....	573
Table 3-722. Function <code>spi_disable</code> .....	574
Table 3-723. Function <code>i2s_init</code> .....	574
Table 3-724. Function <code>i2s_psc_config</code> .....	575
Table 3-725. Function <code>i2s_enable</code> .....	577
Table 3-726. Function <code>i2s_disable</code> .....	577
Table 3-727. Function <code>spi_nss_output_enable</code> .....	578
Table 3-728. Function <code>spi_nss_output_disable</code> .....	578
Table 3-729. Function <code>spi_nss_internal_high</code> .....	579
Table 3-730. Function <code>spi_nss_internal_low</code> .....	579
Table 3-731. Function <code>spi_dma_enable</code> .....	580
Table 3-732. Function <code>spi_dma_disable</code> .....	580
Table 3-733. Function <code>spi_i2s_data_frame_format_config</code> .....	581
Table 3-734. Function <code>spi_i2s_data_transmit</code> .....	582
Table 3-735. Function <code>spi_i2s_data_receive</code> .....	582
Table 3-736. Function <code>spi_bidirectional_transfer_config</code> .....	583
Table 3-737. Function <code>spi_crc_polynomial_set</code> .....	585
Table 3-738. Function <code>spi_crc_polynomial_get</code> .....	586
Table 3-739. Function <code>spi_crc_on</code> .....	586
Table 3-740. Function <code>spi_crc_off</code> .....	587
Table 3-741. Function <code>spi_crc_next</code> .....	587
Table 3-742. Function <code>spi_crc_get</code> .....	588

Table 3-743. Function spi_ti_mode_enable .....	589
Table 3-744. Function spi_ti_mode_disable .....	589
Table 3-745. Function i2s_full_duplex_mode_config .....	583
Table 3-746. Function qspi_enable .....	590
Table 3-747. Function qspi_disable .....	590
Table 3-748. Function qspi_write_enable.....	591
Table 3-749. Function qspi_read_enable .....	591
Table 3-750. Function qspi_io23_output_enable .....	592
Table 3-751. Function qspi_io23_output_disable .....	592
Table 3-752. Function spi_i2s_interrupt_enable .....	594
Table 3-753. Function spi_i2s_interrupt_disable .....	594
Table 3-754. Function spi_i2s_interrupt_flag_get .....	595
Table 3-755. Function spi_i2s_flag_get .....	593
Table 3-756. Function spi_crc_error_clear .....	588
Table 3-757. SYSCFG registers .....	596
Table 3-758.SYSCFG firmware function .....	597
Table 3-759. Function syscfg_deinit .....	597
Table 3-760. Function syscfg_bootmode_config.....	598
Table 3-761. Function syscfg_fmc_swap_config .....	599
Table 3-762. Function syscfg_exmc_swap_config.....	599
Table 3-763. Function syscfg_exti_line_config.....	600
Table 3-764. Function syscfg_enet_phy_interface_config .....	601
Table 3-765. Function syscfg_compensation_config .....	601
Table 3-766. Function syscfg_flag_get.....	602
Table 3-767. TIMERx Registers .....	603
Table 3-768.TIMERx firmware function .....	604
Table 3-769. Structure timer_parameter_struct .....	607
Table 3-770. Structure timer_break_parameter_struct .....	608
Table 3-771. Structure timer_oc_parameter_struct.....	608
Table 3-772. Structure timer_ic_parameter_struct.....	608
Table 3-773. Function timer_deinit.....	609
Table 3-774. Function timer_struct_para_init.....	609
Table 3-775. Function timer_init .....	610
Table 3-776. Function timer_enable .....	611
Table 3-777. Function timer_disable .....	612
Table 3-778. Function timer_auto_reload_shadow_enable .....	612
Table 3-779. Function timer_auto_reload_shadow_disable .....	613
Table 3-780. Function timer_update_event_enable .....	614
Table 3-781. Function timer_update_event_disable .....	614
Table 3-782. Function timer_counter_alignment .....	615
Table 3-783. Function timer_counter_up_direction .....	616
Table 3-784. Function timer_counter_down_direction .....	617
Table 3-785. Function timer_prescaler_config.....	617
Table 3-786. Function timer_repetition_value_config .....	618

Table 3-787. Function timer_autoreload_value_config .....	619
Table 3-788. Function timer_counter_value_config .....	620
Table 3-789. Function timer_counter_read .....	620
Table 3-790. Function timer_prescaler_read .....	621
Table 3-791. Function timer_single_pulse_mode_config .....	622
Table 3-792. Function timer_update_source_config .....	622
Table 3-793. Function timer_dma_enable .....	623
Table 3-794. Function timer_dma_disable .....	624
Table 3-795. Function timer_channel_dma_request_source_select .....	625
Table 3-796. Function timer_dma_transfer_config .....	626
Table 3-797. Function timer_event_software_generate .....	628
Table 3-798. Function timer_break_struct_para_init .....	629
Table 3-799. Function timer_break_config .....	630
Table 3-800. Function timer_break_enable .....	631
Table 3-801. Function timer_break_disable .....	632
Table 3-802. Function timer_automatic_output_enable .....	632
Table 3-803. Function timer_automatic_output_disable .....	633
Table 3-804. Function timer_primary_output_config .....	634
Table 3-805. Function timer_channel_control_shadow_config .....	634
Table 3-806. Function timer_channel_control_shadow_update_config .....	635
Table 3-807. Function timer_channel_output_struct_para_init .....	636
Table 3-808. Function timer_channel_output_config .....	637
Table 3-809. Function timer_channel_output_mode_config .....	638
Table 3-810. Function timer_channel_output_pulse_value_config .....	639
Table 3-811. Function timer_channel_output_shadow_config .....	640
Table 3-812. Function timer_channel_output_fast_config .....	641
Table 3-813. Function timer_channel_output_clear_config .....	642
Table 3-814. Function timer_channel_output_polarity_config .....	644
Table 3-815. Function timer_channel_complementary_output_polarity_config .....	645
Table 3-816. Function timer_channel_output_state_config .....	646
Table 3-817. Function timer_channel_complementary_output_state_config .....	647
Table 3-818. Function timer_channel_input_struct_para_init .....	648
Table 3-819. Function timer_input_capture_config .....	648
Table 3-820. Function timer_channel_input_capture_prescaler_config .....	649
Table 3-821. Function timer_channel_capture_value_register_read .....	651
Table 3-822. Function timer_input_pwm_capture_config .....	652
Table 3-823. Function timer_hall_mode_config .....	653
Table 3-824. Function timer_input_trigger_source_select .....	653
Table 3-825. Function timer_master_output_trigger_source_select .....	655
Table 3-826. Function timer_slave_mode_select .....	656
Table 3-827. Function timer_master_slave_mode_config .....	657
Table 3-828. Function timer_external_trigger_config .....	658
Table 3-829. Function timer_quadrature_decoder_mode_config .....	659
Table 3-830. Function timer_internal_clock_config .....	660

Table 3-831. Function timer_internal_trigger_as_external_clock_config .....	661
Table 3-832. Function timer_external_trigger_as_external_clock_config .....	662
Table 3-833. Function timer_external_clock_mode0_config .....	663
Table 3-834. Function timer_external_clock_mode1_config .....	664
Table 3-835. Function timer_external_clock_mode1_disable .....	666
Table 3-836. Function timer_channel_remap_config .....	666
Table 3-837. Function timer_write_chxval_register_config .....	668
Table 3-838. Function timer_output_value_selection_config .....	668
Table 3-839. Function timer_flag_get .....	669
Table 3-840. Function timer_flag_clear .....	670
Table 3-841. Function timer_interrupt_enable .....	672
Table 3-842. Function timer_interrupt_disable .....	673
Table 3-843. Function timer_interrupt_flag_get .....	674
Table 3-844. Function timer_interrupt_flag_clear .....	675
Table 3-845. TLI Registers .....	676
Table 3-846. TLI firmware function .....	677
Table 3-847. tli_parameter_struct .....	678
Table 3-848. tli_layer_parameter_struct .....	679
Table 3-849. tli_layer_lut_parameter_struct .....	680
Table 3-850. Function tli_deinit .....	680
Table 3-851. Function tli_struct_para_init .....	680
Table 3-852. Function tli_init .....	681
Table 3-853. Function tli_dither_config .....	682
Table 3-854. Function tli_enable .....	683
Table 3-855. Function tli_disable .....	684
Table 3-856. Function tli_reload_config .....	684
Table 3-857. Function tli_layer_struct_para_init .....	685
Table 3-858. Function tli_layer_init .....	686
Table 3-859. Function tli_layer_window_offset_modify .....	687
Table 3-860. Function tli_lut_struct_para_init .....	688
Table 3-861. Function tli_lut_init .....	689
Table 3-862. Function tli_color_key_init .....	690
Table 3-863. Function tli_layer_enable .....	690
Table 3-864. Function tli_layer_disable .....	691
Table 3-865. Function tli_color_key_enable .....	692
Table 3-866. Function tli_color_key_disable .....	692
Table 3-867. Function tli_lut_enable .....	693
Table 3-868. Function tli_lut_disable .....	694
Table 3-869. Function tli_line_mark_set .....	694
Table 3-870. Function tli_current_pos_get .....	695
Table 3-871. Function tli_interrupt_enable .....	695
Table 3-872. Function tli_interrupt_disable .....	696
Table 3-873. Function tli_interrupt_flag_get .....	697
Table 3-874. Function tli_interrupt_flag_clear .....	698

Table 3-875. Function tli_flag_get .....	699
Table 3-876 TRNG Registers .....	700
Table 3-877. TRNG firmware function .....	700
Table 3-878. Function trng_deinit.....	700
Table 3-879. Function trng_enable .....	701
Table 3-880 Function trng_disable.....	702
Table 3-881 Function trng_get_true_random_data .....	702
Table 3-882 Function trng_interrupt_enable .....	703
Table 3-883 Function trng_interrupt_disable .....	703
Table 3-884 Function trng_flag_get.....	704
Table 3-885 Function trng_interrupt_flag_get .....	705
Table 3-886 Function trng_interrupt_flag_clear .....	706
Table 3-887. USART Registers .....	706
Table 3-888. USART firmware function.....	707
Table 3-889. Function usart_deinit.....	708
Table 3-890. Function usart_baudrate_set .....	709
Table 3-891. Function usart_parity_config .....	709
Table 3-892. Function usart_word_length_set.....	710
Table 3-893. Function usart_stop_bit_set.....	711
Table 3-894. Function usart_enable .....	711
Table 3-895. Function usart_disable .....	712
Table 3-896. Function usart_transmit_config.....	712
Table 3-897. Function usart_receive_config .....	713
Table 3-898. Function usart_data_first_config .....	714
Table 3-899. Function usart_invert_config .....	714
Table 3-900. Function usart_oversample_config .....	715
Table 3-901. Function usart_sample_bit_config.....	716
Table 3-902. Function usart_receiver_timeout_enable.....	716
Table 3-903. Function usart_receiver_timeout_disable.....	717
Table 3-904. Function usart_receiver_timeout_threshold_config .....	717
Table 3-905. Function usart_data_transmit .....	718
Table 3-906. Function usart_data_receive .....	719
Table 3-907. Function usart_address_config .....	719
Table 3-908. Function usart_mute_mode_enable.....	720
Table 3-909. Function usart_mute_mode_disable.....	720
Table 3-910. Function usart_mute_mode_wakeup_config .....	721
Table 3-911. Function usart_lin_mode_enable .....	721
Table 3-912. Function usart_lin_mode_disable .....	722
Table 3-913. Function usart_lin_break_dection_length_config .....	722
Table 3-914. Function usart_send_break .....	723
Table 3-915. Function usart_halfduplex_enable .....	724
Table 3-916. Function usart_halfduplex_disable .....	724
Table 3-917. Function usart_synchronous_clock_enable .....	725
Table 3-918. Function usart_synchronous_clock_disable .....	725

Table 3-919. Function <code>usart_synchronous_clock_config</code> .....	726
Table 3-920. Function <code>usart_guard_time_config</code> .....	727
Table 3-921. Function <code>usart_smartcard_mode_enable</code> .....	727
Table 3-922. Function <code>usart_smartcard_mode_disable</code> .....	728
Table 3-923. Function <code>usart_smartcard_mode_nack_enable</code> .....	728
Table 3-924. Function <code>usart_smartcard_mode_nack_disable</code> .....	729
Table 3-925. Function <code>usart_smartcard_autoretry_config</code> .....	729
Table 3-926. Function <code>usart_block_length_config</code> .....	730
Table 3-927. Function <code>usart_irda_mode_enable</code> .....	730
Table 3-928. Function <code>usart_irda_mode_disable</code> .....	731
Table 3-929. Function <code>usart_prescaler_config</code> .....	731
Table 3-930. Function <code>usart_irda_lowpower_config</code> .....	732
Table 3-931. Function <code>usart_hardware_flow_rts_config</code> .....	733
Table 3-932. Function <code>usart_hardware_flow_cts_config</code> .....	733
Table 3-933. Function <code>usart_break_frame_coherence_config</code> .....	734
Table 3-934. Function <code>usart_parity_check_coherence_config</code> .....	734
Table 3-935. Function <code>usart_hardware_flow_coherence_config</code> .....	735
Table 3-936. Function <code>usart_dma_receive_config</code> .....	736
Table 3-937. Function <code>usart_dma_transmit_config</code> .....	736
Table 3-938. Function <code>usart_flag_get</code> .....	737
Table 3-939. Function <code>usart_flag_clear</code> .....	738
Table 3-940. Function <code>usart_interrupt_enable</code> .....	739
Table 3-941. Function <code>usart_interrupt_disable</code> .....	739
Table 3-942. Function <code>usart_interrupt_flag_get</code> .....	740
Table 3-943. Function <code>usart_interrupt_flag_clear</code> .....	741
Table 3-944. WWDGT Registers .....	742
Table 3-945. WWDGT firmware function .....	743
Table 3-946. Function <code>wwdgt_deinit</code> .....	743
Table 3-947. Function <code>wwdgt_enable</code> .....	744
Table 3-948. Function <code>wwdgt_counter_update</code> .....	744
Table 3-949. Function <code>wwdgt_config</code> .....	745
Table 3-950. Function <code>wwdgt_interrupt_enable</code> .....	746
Table 3-951. Function <code>wwdgt_flag_get</code> .....	746
Table 3-952. Function <code>wwdgt_flag_clear</code> .....	747
Table 4-1. Revision history .....	749



## 1. Introduction

This manual introduces firmware library of GD32A490 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 GD32A490 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
CAN	Controller area network
CRC	CRC calculation unit
CTC	Clock trim controller
DAC	Digital-to-analog converter
DBG	Debug



Peripherals	Descriptions
DCI	Digital camera interface
DMA	Direct memory access controller
ENET	Ethernet
EXMC	External memory controller
EXTI	Interrupt/event controller
FMC	Flash memory controller
FWDGT	Free watchdog timer
GPIO/AFIO	General-purpose and alternate-function I/Os
I2C	Inter-integrated circuit interface
IPA	Image processing accelerator
IREF	Programmable current reference
MISC	Nested Vectored Interrupt Controller
PMU	Power management unit
RCU	Reset and clock unit
RTC	Real-time Clock
SDIO	Secure digital input/output interface
SPI/I2S	Serial peripheral interface/Inter-IC sound
SYSCFG	System configuration
TIMER	TIMER
TLI	TFT-LCD interface
TRNG	True random number generator
USART	Universal synchronous/asynchronous receiver /transmitter
WWDGT	Window watchdog timer
USBFS	Universal serial bus full-speed interface

### 1.1.2. Naming rules

The firmware library naming rules are shown as below:

- The peripherals are shortened in XXX format, such as: ADC. More shorten information of peripherals refer to [Peripherals](#);
- The name of sourcefile and header file are started with “gd32a490\_”, such as: gd32a490\_adc.h;
- The constants used only in one file should be defined in the used file; the constants used in many files should be defined in corresponding header file. All the constants are written in uppercase of English letters;
- Registers are handled as constants. The naming of them are written in uppercase of English letters. In most cases, register names are shortened accord with the user manual;
- Variables are written in lowercase, when concluded by several words, underlines should be adapted among words;
- The naming of peripheral functions are started with the peripheral abbreviation added with an underline, when the function name is concluded by several words, underlines should be adapted among words, and all the peripheral functions are written in lowercase.

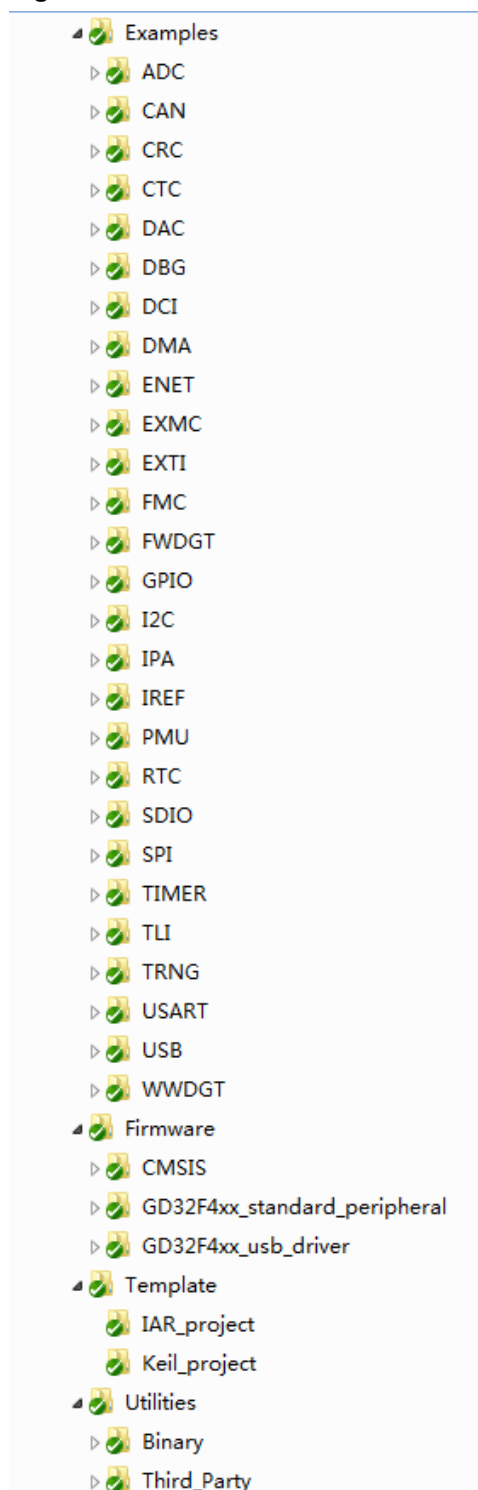


## 2. Firmware Library Overview

### 2.1. File Structure of Firmware Library

GD32A490\_Firmware\_Library, the file structure is shown as below:

**Figure 2-1. File structure of firmware library of GD32A490**



### 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;
- gd32a490\_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);
- gd32a490\_it.c: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- gd32a490\_it.h: the header file include all the prototypes of the interrupt service routines;
- systick.c: the source file include the precise time delay functions by using systick;
- systick.h: the header file include the prototype of the precise time delay functions by using systick;
- main.c: example code. Note: all the examples are not influenced by software IDEs.

### 2.1.2. Firmware Folder

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

- CMSIS subfolder includes the Cortex M4 kernel support files, the startup file based on the Cortex M4 kernel processor, the global header file of GD32A490 and system configuration file;
- GD32A490\_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;
- GD32A490\_usbfs\_driver subfolder includes all the related files about USBFS peripheral:
  - Include subfolder includes the header files of USBFS peripheral, users need not modify this folder;
  - Source subfolder includes the source files of USBFS peripheral, users need not modify this folder;

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

### 2.1.3. Project Folder

Project folder includes examples of USBFS (EWARM is run in IAR, and MDK-ARM is run in Keil4).

### 2.1.4. Template Folder

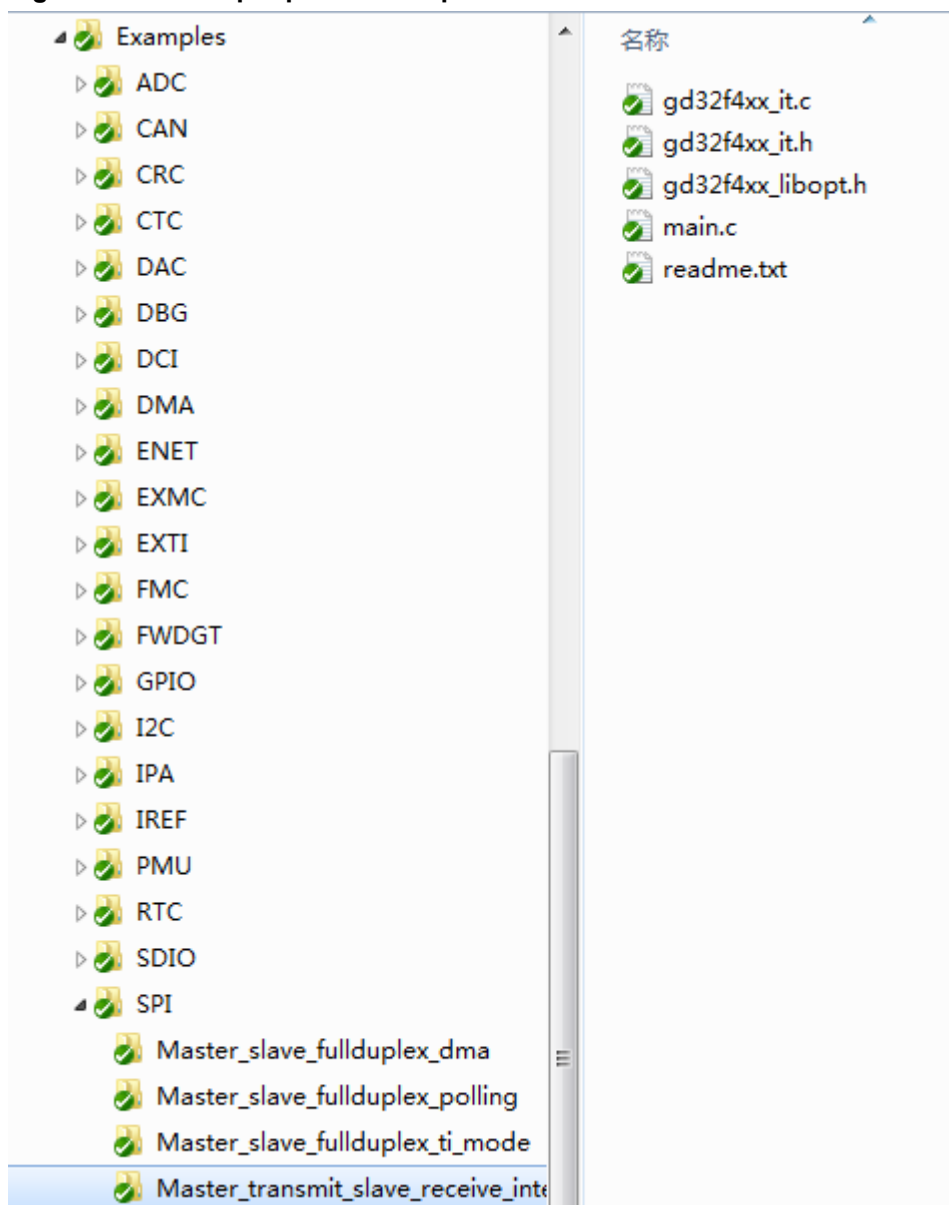
Template folder includes a simple demo of how to use LED, how to print by USART and use

key to control, (IAR\_project is run in IAR, and Keil\_project is run in Keil4). User can use the project template to compile the formware examples, the steps are shown as below:

### Select files

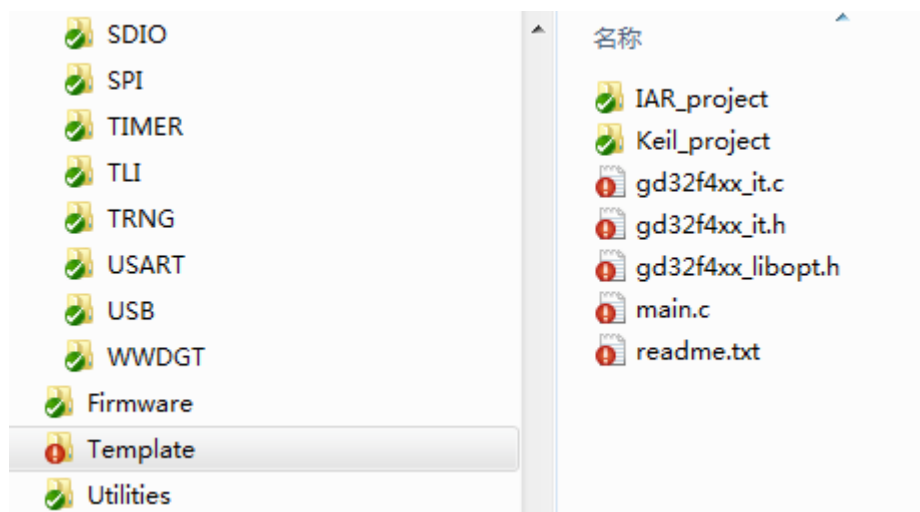
Open “Examples” folder, select the module to be tested, such as SPI, open ”SPI” folder, select an example of SPI, such as ”SPI\_master\_transmit\_slave\_receive\_interrupt”, shown as below:

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



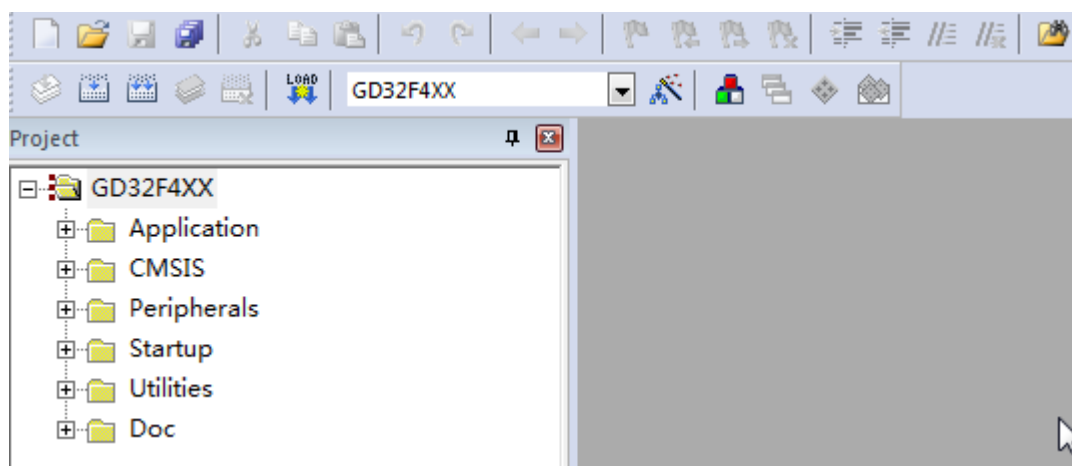
### Copy files

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

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

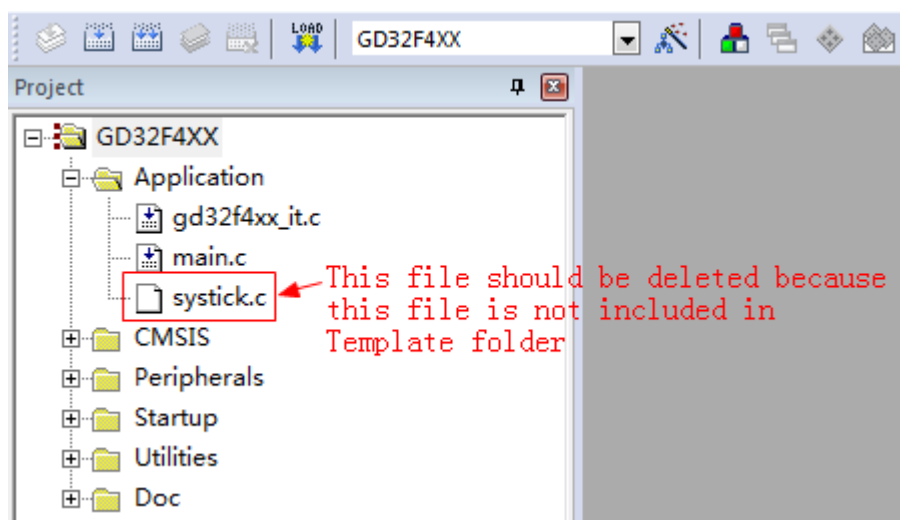
### Open a project

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

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

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

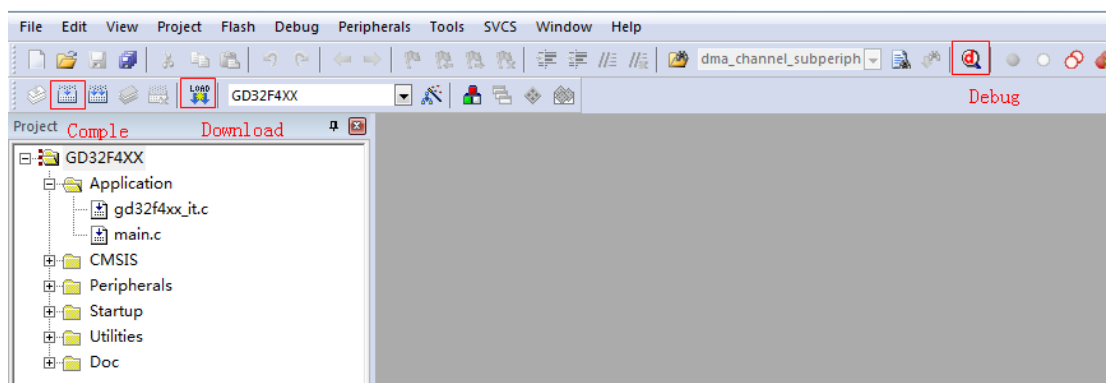
Figure 2-5. Configure project files



### Compile-Debug-Download

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

Figure 2-6. Compile-debug-download



#### 2.1.5. Utilities Folder

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

- Binary, Third\_Party subfolders include files for USB tests;
- gd32a490\_eval.h is related header files of the evaluation board about running the firmware examples;
- gd32a490\_eval.c is related source files of the evaluation board about running the firmware examples.

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

## 2.2. File descriptions of Firmware Library

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

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

Files	Descriptions
gd32a490_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.
gd32a490_it.h	Header file, including all the prototypes of interrupt service routines.
gd32a490_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.
gd32a490_xxx.h	The header file of peripheral PPP, including functions about peripheral PPP, and the variables used for functions.
gd32a490_xxx.c	The C source file for driving peripheral PPP.
systick.h	The header file of systick.c, including prototypes of systick configuration function and delay function.
systick.c	The source file about systick configuration function and delay function.
readme.txt	Description document about how to configure and how to use the firmware example.



## 3. Firmware Library of Standard Peripherals

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

The description format of firmware functions are shown as below:

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

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

### 3.2. ADC

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

#### 3.2.1. Descriptions of Peripheral registers

ADC registers are listed in the table shown as below:

**Table 3-2. ADC Registers**

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

Registers	Descriptions
ADC_RSQ0	Routine sequence register 0
ADC_RSQ1	Routine sequence register 1
ADC_RSQ2	Routine sequence register 2
ADC_ISQ	Inserted sequence register
ADC_IDATAx	Inserted data register x(x=0..3)
ADC_RDATA	Routine data register
ADC_OVSAMPCTL	Oversample control register
ADC_SSTAT	Summary status register
ADC_SYNCCTL	Sync control register
ADC_SYNCDATA	Sync routine data register

### 3.2.2. Descriptions of Peripheral functions

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

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

Function name	Function description
adc_deinit	reset ADCx peripheral
adc_clock_config	configure the ADC clock for all the ADCs
adc_special_function_config	enable or disable ADC special function
adc_data_alignment_config	configure ADC data alignment
adc_enable	enable ADC interface
adc_disable	disable ADC interface
adc_calibration_enable	ADC calibration and reset calibration
adc_channel_16_to_18	configure temperature sensor and internal reference voltage channel or VBAT channel function
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_dma_mode_enable	enable DMA request
adc_dma_mode_disable	disable DMA request
adc_dma_request_after_last_enable	when DMA=1, the DMA engine issues a request at end of each routine conversion
adc_dma_request_after_last_disable	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
adc_discontinuous_mode_config	configure ADC discontinuous mode
adc_channel_length_config	configure the length of routine channel group or inserted channel group
adc_routine_channel_config	configure ADC routine channel
adc_inserted_channel_config	configure ADC inserted channel
adc_inserted_channel_offset_config	configure ADC inserted channel offset

Function name	Function description
adc_external_trigger_source_config	configure ADC external trigger source
adc_external_trigger_config	enable ADC external trigger
adc_software_trigger_enable	enable ADC software trigger
adc_end_of_conversion_config	configure end of conversion mode
adc_routine_data_read	read ADC routine group data register
adc_inserted_data_read	read ADC inserted group data register
adc_watchdog_single_channel_disable	disable ADC analog watchdog single channel
adc_watchdog_single_channel_enable	configure ADC analog watchdog single channel
adc_watchdog_group_channel_enable	configure ADC analog watchdog group channel
adc_watchdog_disable	disable ADC analog watchdog
adc_watchdog_threshold_config	configure ADC analog watchdog threshold
adc_flag_get	get the ADC flag bits
adc_flag_clear	clear the ADC flag bits
adc_routine_software_startconv_flag_get	get the bit state of ADCx software start conversion
adc_inserted_software_startconv_flag_get	get the bit state of ADCx software inserted channel start conversion
adc_interrupt_flag_get	get the ADC interrupt bits
adc_interrupt_flag_clear	clear the ADC flag
adc_interrupt_enable	enable ADC interrupt
adc_interrupt_disable	disable ADC interrupt
adc_sync_mode_config	configure the ADC sync mode
adc_sync_delay_config	configure the delay between 2 sampling phases in ADC sync modes
adc_sync_dma_config	configure ADC sync DMA mode selection
adc_sync_dma_request_after_last_enable	when SYNCDMA is not equal to 2'b00, the DMA engine issues requests according to the SYNCDMA bits
adc_sync_dma_request_after_last_disable	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
adc_sync_routine_data_read	read ADC sync routine data register

## adc\_deinit

The description of adc\_deinit is shown as below:

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

Function name	adc_deinit
Function prototype	void adc_deinit(void);
Function descriptions	reset ADC peripheral
Precondition	-

<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 ADC */
```

```
adc_deinit();
```

### adc\_clock\_config

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

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

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

<i>IV10</i>	
<i>ADC_ADCCCK_HCLK_D</i> <i>IV20</i>	HCLK div20
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the ADC clock: PCLK2 div2 */
adc_clock_config (ADC_ADCCCK_PCLK2_DIV2);
```

### adc\_special\_function\_config

The description of `adc_special_function_config` is shown as below:

**Table 3-6. Function `adc_special_function_config`**

<b>Function name</b>	<code>adc_special_function_config</code>
<b>Function prototype</b>	<code>void adc_special_function_config(uint32_t adc_periph, uint32_t function, ControlStatus newvalue);</code>
<b>Function descriptions</b>	enable or disable ADC special function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>function</b>	the function to config
<i>ADC_SCAN_MODE</i>	scan mode select
<i>ADC_INSERTED_CHANNELNEL_AUTO</i>	inserted channel group convert automatically
<i>ADC_CONTINUOUS_MODE</i>	continuous mode select
<b>Input parameter{in}</b>	

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

Example:

```
/* enable ADC0 scan mode */
```

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

### adc\_data\_alignment\_config

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

**Table 3-7. Function adc\_data\_alignment\_config**

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

-	-
Return value	
-	-

Example:

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

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

### adc\_enable

The description of adc\_enable is shown as below:

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

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

Example:

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

```
adc_enable(ADC0);
```

### adc\_disable

The description of adc\_disable is shown as below:

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

Function name	adc_disable
---------------	-------------

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

Example:

```
/* disable ADC0 */
```

```
adc_disable(ADC0);
```

### adc\_calibration\_enable

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

**Table 3-10. Function adc\_calibration\_enable**

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



-	-
---	---

Example:

```
/* ADC0 calibration and reset calibration */
```

```
adc_calibration_enable(ADC0);
```

### adc\_channel\_16\_to\_18

The description of adc\_channel\_16\_to\_18 is shown as below:

**Table 3-11. Function adc\_channel\_16\_to\_18**

<b>Function name</b>	adc_channel_16_to_18
<b>Function prototype</b>	void adc_channel_16_to_18(uint32_t function, ControlStatus newvalue);
<b>Function descriptions</b>	configure temperature sensor and internal reference voltage channel or VBAT channel function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>function</b>	temperature sensor and internal reference voltage channel or VBAT channel
ADC_VBAT_CHANNEL_SWITCH	channel 18 (1/4 voltate of external battery) switch of ADC0
ADC_TEMP_VREF_CHANNEL_SWITCH	channel 16 (temperature sensor) and 17 (internal reference voltage) switch of ADC0
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
ENABLE	enable function
DISABLE	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_channel_16_to_18 (ADC_TEMP_VREF_CHANNEL_SWITCH, ENABLE);
```

### adc\_resolution\_config

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

**Table 3-12. 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>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC0 resolution: 10 bits */
```

```
adc_resolution_config (ADC0, ADC_RESOLUTION_10B);
```

## adc\_oversample\_mode\_config

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

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

<b>Function name</b>	adc_oversample_mode_config
<b>Function prototype</b>	void adc_oversample_mode_config(uint32_t adc_periph, uint32_t mode, uint16_t shift, uint8_t ratio);
<b>Function descriptions</b>	configure ADC oversample mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>mode</b>	ADC oversampling mode
ADC_OVERSAMPLING _ALL_CONVERT	all oversampled conversions for a channel are done consecutively after a trigger
ADC_OVERSAMPLING _ONE_CONVERT	each oversampled conversion for a channel needs a trigger
<b>Input parameter{in}</b>	
<b>shift</b>	ADC oversampling shift
ADC_OVERSAMPLING _SHIFT_NONE	no oversampling shift
ADC_OVERSAMPLING _SHIFT_1B	1-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_2B	2-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_3B	3-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_4B	4-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_5B	5-bit oversampling shift

ADC_OVERSAMPLING_SHIFT_6B	6-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_7B	7-bit oversampling shift
ADC_OVERSAMPLING_SHIFT_8B	8-bit oversampling shift
<b>Input parameter{in}</b>	
<b>ratio</b>	ADC oversampling ratio
ADC_OVERSAMPLING_RATIO_MUL2	oversampling ratio multiple 2
ADC_OVERSAMPLING_RATIO_MUL4	oversampling ratio multiple 4
ADC_OVERSAMPLING_RATIO_MUL8	oversampling ratio multiple 8
ADC_OVERSAMPLING_RATIO_MUL16	oversampling ratio multiple 16
ADC_OVERSAMPLING_RATIO_MUL32	oversampling ratio multiple 32
ADC_OVERSAMPLING_RATIO_MUL64	oversampling ratio multiple 64
ADC_OVERSAMPLING_RATIO_MUL128	oversampling ratio multiple 128
ADC_OVERSAMPLING_RATIO_MUL256	oversampling ratio multiple 256
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## adc\_oversample\_mode\_enable

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

**Table 3-14. 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>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC0 oversample mode */
adc_oversample_mode_enable (ADC0);
```

## adc\_oversample\_mode\_disable

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

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

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

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_oversample_mode_disable (ADC0);
```

### adc\_dma\_mode\_enable

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

**Table 3-16. Function adc\_dma\_mode\_enable**

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

Example:

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

```
adc_dma_mode_enable(ADC0);
```

### adc\_dma\_mode\_disable

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

Table 3-17. Function `adc_dma_mode_disable`

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

Example:

```
/* disable ADC0 DMA request */
adc_dma_mode_disable(ADC0);
```

### `adc_dma_request_after_last_enable`

The description of `adc_dma_request_after_last_enable` is shown as below:

Table 3-18. Function `adc_dma_request_after_last_enable`

Function name	<code>adc_dma_request_after_last_enable</code>
Function prototype	<code>void adc_dma_request_after_last_enable(uint32_t adc_periph);</code>
Function descriptions	when DMA=1, the DMA engine issues a request at end of each routine conversion
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC peripheral
<code>ADCx(x=0..2)</code>	ADC peripheral selection
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* when DMA=1, the DMA engine issues a request at end of each routine conversion for ADC0
*/
```

```
adc_dma_request_after_last_enable (ADC0);
```

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

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

**Table 3-19. Function adc\_dma\_mode\_disable**

<b>Function name</b>	adc_dma_request_after_last_disable
<b>Function prototype</b>	void adc_dma_request_after_last_disable(uint32_t adc_periph);
<b>Function descriptions</b>	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* the DMA engine is disabled after the end of transfer signal from DMA controller is detected
for ADC0 */
```

```
adc_dma_request_after_last_disable (ADC0);
```

### adc\_discontinuous\_mode\_config

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



Table 3-20. Function `adc_discontinuous_mode_config`

<b>Function name</b>	<code>adc_discontinuous_mode_config</code>
<b>Function prototype</b>	<code>void adc_discontinuous_mode_config(uint32_t adc_periph, uint8_t adc_channel_group, uint8_t length);</code>
<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
<code>ADCx(x=0..2)</code>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_channel_group</b>	select the channel group
<code>ADC_ROUTINE_CHANNEL</code>	routine channel group
<code>ADC_INSERTED_CHANNEL</code>	inserted channel group
<code>ADC_CHANNEL_DISCON_DISABLE</code>	disable discontinuous mode of routine and inserted channel
<b>Input parameter{in}</b>	
<b>length</b>	number of conversions in discontinuous mode, the number can be 1..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 channel group discontinuous mode */
adc_discontinuous_mode_config(ADC0, ADC_ROUTINE_CHANNEL, 6);
```

### **adc\_channel\_length\_config**

The description of `adc_channel_length_config` is shown as below:

**Table 3-21. Function adc\_channel\_length\_config**

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

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 3-22. 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
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>rank</b>	the routine group sequence rank, this parameter must be between 0 to 15
<b>Input parameter{in}</b>	
<b>adc_channel</b>	the selected ADC channel
<i>ADC_CHANNEL_x(x=0..18)</i>	ADC Channelx (x=0..17)(x=16..18 are only for ADC0)
<b>Input parameter{in}</b>	
<b>sample_time</b>	the sample time value
<i>ADC_SAMPLETIME_3</i>	3cycles
<i>ADC_SAMPLETIME_1</i> 5	15cycles
<i>ADC_SAMPLETIME_2</i> 8	28 cycles
<i>ADC_SAMPLETIME_5</i> 6	56 cycles
<i>ADC_SAMPLETIME_8</i> 4	84cycles
<i>ADC_SAMPLETIME_1</i> 12	112 cycles
<i>ADC_SAMPLETIME_1</i> 44	144cycles
<i>ADC_SAMPLETIME_4</i> 80	480 cycles
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

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

```
adc_routine_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_56);
```

### adc\_inserted\_channel\_config

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

**Table 3-23. Function adc\_inserted\_channel\_config**

Function name	adc_inserted_channel_config
Function prototype	void adc_inserted_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
Function descriptions	configure ADC inserted channel
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
rank	the inserted group sequencer rank, this parameter must be between 0 to 3
Input parameter{in}	
adc_channel	the selected ADC channel
ADC_CHANNEL_x(x=0..18)	ADC Channelx (x=0..17)(x=16..18 are only for ADC0)
Input parameter{in}	
sample_time	the sample time value
ADC_SAMPLETIME_3	3 cycles
ADC_SAMPLETIME_15	15 cycles
ADC_SAMPLETIME_28	28 cycles

ADC_SAMPLETIME_5 6	56 cycles
ADC_SAMPLETIME_8 4	48 cycles
ADC_SAMPLETIME_1 12	112 cycles
ADC_SAMPLETIME_1 44	144 cycles
ADC_SAMPLETIME_4 80	480 cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_inserted_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_56);
```

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

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

**Table 3-24. Function adc\_inserted\_channel\_offset\_config**

Function name	adc_inserted_channel_offset_config
Function prototype	void adc_inserted_channel_offset_config(uint32_t adc_periph, uint8_t inserted_channel, uint16_t offset);
Function descriptions	configure ADC inserted channel offset
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	

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

Example:

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

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

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

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

**Table 3-25. Function adc\_external\_trigger\_source\_config**

<b>Function name</b>	adc_external_trigger_source_config
<b>Function prototype</b>	void adc_external_trigger_source_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t external_trigger_source);
<b>Function descriptions</b>	configure ADC external trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_channel_group</b>	select the channel group
<i>ADC_ROUTINE_CHANNEL</i>	routine channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group

Input parameter{in}	
<b>external_trigger_source</b>	routine or inserted group trigger source
ADC_EXTTRIG_ROUTINE_T0_CH0	external trigger timer 0 CC0 event select for routine channel
ADC_EXTTRIG_ROUTINE_T0_CH1	external trigger timer 0 CC1 event select for routine channel
ADC_EXTTRIG_ROUTINE_T0_CH2	external trigger timer 7 CC2 event select for routine channel
ADC_EXTTRIG_ROUTINE_T1_CH1	external trigger timer 1 CC1 event select for routine channel
ADC_EXTTRIG_ROUTINE_T1_CH2	external trigger timer 1 CC2 event select for routine channel
ADC_EXTTRIG_ROUTINE_T1_CH3	external trigger timer 1 CC3 event select for routine channel
ADC_EXTTRIG_ROUTINE_T1_TRGO	external trigger timer 1 TRGO event select for routine channel
ADC_EXTTRIG_ROUTINE_T2_CH0	external trigger timer 2 CC0 event select for routine channel
ADC_EXTTRIG_ROUTINE_T2_TRGO	external trigger timer 2 TRGO event select for routine channel
ADC_EXTTRIG_ROUTINE_T3_CH3	external trigger timer 3 CC3 event select for routine channel
ADC_EXTTRIG_ROUTINE_T4_CH0	external trigger timer 4 CC0 event select for routine channel
ADC_EXTTRIG_ROUTINE_T4_CH1	external trigger timer 4 CC1 event select for routine channel
ADC_EXTTRIG_ROUTINE_T4_CH2	external trigger timer 4 CC2 event select for routine channel
ADC_EXTTRIG_ROUTINE_T7_CH0	external trigger timer 7 CC0 event select for routine channel
ADC_EXTTRIG_ROUTINE_T7_TRGO	external trigger timer 7 TRGO event select for routine channel
ADC_EXTTRIG_ROUTINE_EXTI_11	external trigger extiline 11 select for routine channel

ADC_EXTTRIG_INSERTED_T0_CH3	external trigger timer 0 CH3 event select for inserted channel
ADC_EXTTRIG_INSERTED_T0_TRGO	external trigger timer 0 TRGO event select for inserted channel
ADC_EXTTRIG_INSERTED_T1_CH0	external trigger timer 1 CH0 event select for inserted channel
ADC_EXTTRIG_INSERTED_T1_TRGO	external trigger timer 1 TRGO event select for inserted channel
ADC_EXTTRIG_INSERTED_T2_CH1	external trigger timer 2 CH1 event select for inserted channel
ADC_EXTTRIG_INSERTED_T2_CH3	external trigger timer 2 CH3 event select for inserted channel
ADC_EXTTRIG_INSERTED_T3_CH0	external trigger timer 3 CH0 event select for inserted channel
ADC_EXTTRIG_INSERTED_T3_CH1	external trigger timer 3 CH1 event select for inserted channel
ADC_EXTTRIG_INSERTED_T3_CH2	external trigger timer 3 CH2 event select for inserted channel
ADC_EXTTRIG_INSERTED_T3_TRGO	external trigger timer 3 TRGO event select for inserted channel
ADC_EXTTRIG_INSERTED_T4_CH3	external trigger timer 4 CH3 event select for inserted channel
ADC_EXTTRIG_INSERTED_T4_TRGO	external trigger timer 4 TRGO event select for inserted channel
ADC_EXTTRIG_INSERTED_T7_CH1	external trigger timer 7 CH1 event select for inserted channel
ADC_EXTTRIG_INSERTED_T7_CH2	external trigger timer 7 CH2 event select for inserted channel
ADC_EXTTRIG_INSERTED_T7_CH3	external trigger timer 7 CH3 event select for inserted channel
ADC_EXTTRIG_INSERTED_EXTI_15	external trigger extiline 15 select for inserted channel
Output parameter{out}	
-	-



Return value	
-	-

Example:

```
/* configure ADC0 routine channel external trigger source */
```

```
adc_external_trigger_source_config(ADC0,ADC_ROUTINE_CHANNEL,
ADC_EXTTRIG_ROUTINE_T0_CH0);
```

### adc\_external\_trigger\_config

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

**Table 3-26. Function adc\_external\_trigger\_config**

Function name	adc_external_trigger_config
Function prototype	void adc_external_trigger_config(uint32_t adc_periph , uint8_t adc_channel_group , uint32_t trigger_mode);
Function descriptions	configure ADC external trigger
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_channel_group	select the channel group
ADC_ROUTINE_CHANNEL	routine channel group
ADC_INSERTED_CHANNEL	inserted channel group
Input parameter{in}	
trigger_mode	external trigger mode
EXTERNAL_TRIGGER_DISABLE	external trigger disable
EXTERNAL_TRIGGER_RISING	rising edge of external trigger

<i>EXTERNAL_TRIGGER_FALLING</i>	falling edge of external trigger
<i>EXTERNAL_TRIGGER_RISING_FALLING</i>	rising and falling edge of external trigger
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### adc\_software\_trigger\_enable

The description of `adc_software_trigger_enable` is shown as below:

**Table 3-27. Function `adc_software_trigger_enable`**

<b>Function name</b>	<code>adc_software_trigger_enable</code>
<b>Function prototype</b>	<code>void adc_software_trigger_enable(uint32_t adc_periph, uint8_t adc_channel_group);</code>
<b>Function descriptions</b>	enable ADC software trigger
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_channel_group</b>	select the channel group
<i>ADC_ROUTINE_CHANNEL</i>	routine channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC0 routine channel group software trigger */
```

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

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

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

**Table 3-28. Function adc\_end\_of\_conversion\_config**

Function name	adc_end_of_conversion_config
Function prototype	void adc_end_of_conversion_config(uint32_t adc_periph , uint8_t end_selection);
Function descriptions	configure end of conversion mode
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
end_selection	end of conversion mode
ADC_EOC_SET_SEQ UENCE	only at the end of a sequence of routine conversions, the EOC bit is set.Overflow detection is disabled unless DMA=1
ADC_EOC_SET_CON VERSION	at the end of each routine conversion, the EOC bit is set.Overflow is detected automatically
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* at the end of each routine conversion, the EOC bit is set. */
```

```
adc_end_of_conversion_config (ADC0, ADC_EOC_SET_CONVERSION);
```

### adc\_routine\_data\_read

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

**Table 3-29. 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 group data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	ADC conversion value (0-0xFFFF)

Example:

```
/* read ADC0 routine group 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 3-30. Function adc\_inserted\_data\_read**

<b>Function name</b>	adc_inserted_data_read
<b>Function prototype</b>	uint16_t adc_inserted_data_read(uint32_t adc_periph, uint8_t inserted_channel);
<b>Function descriptions</b>	read ADC inserted group data register
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>inserted_channel</b>	insert channel select
<i>ADC_INSERTED_CHANNEL_x(x=0..3)</i>	inserted Channelx, x=0,1,2,3
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	ADC conversion value (0-0xFFFF)

Example:

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

```
uint16_t adc_value = 0;
```

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

### adc\_watchdog\_single\_channel\_disable

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

**Table 3-31. Function adc\_watchdog\_single\_channel\_disable**

<b>Function name</b>	adc_watchdog_single_channel_disable
<b>Function prototype</b>	void adc_watchdog_single_channel_disable(uint32_t adc_periph );
<b>Function descriptions</b>	disable ADC analog watchdog 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..2)</i>	ADC peripheral selection
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* disable ADC0 analog watchdog single channel */
```

```
adc_watchdog_single_channel_disable(ADC0);
```

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

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

**Table 3-32. Function adc\_watchdog\_single\_channel\_enable**

Function name	adc_watchdog_single_channel_enable
Function prototype	void adc_watchdog_single_channel_enable(uint32_t adc_periph, uint8_t adc_channel);
Function descriptions	configure ADC analog watchdog single channel
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_channel	the selected ADC channel
ADC_CHANNEL_x(x=0..17)	ADC Channelx(x=0..17) (x=16 and x=17 are only for ADC0)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_watchdog_single_channel_enable(ADC0, ADC_CHANNEL_1);
```

## adc\_watchdog\_group\_channel\_enable

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

**Table 3-33. Function adc\_watchdog\_group\_channel\_enable**

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

Example:

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

## adc\_watchdog\_disable

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

Table 3-34. Function `adc_watchdog_disable`

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

Example:

```
/* disable ADC0 analog watchdog */
adc_watchdog_disable(ADC0);
```

### `adc_watchdog_threshold_config`

The description of `adc_watchdog_threshold_config` is shown as below:

Table 3-35. Function `adc_watchdog_threshold_config`

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



<b>low_threshold</b>	analog watchdog low threshold, 0..4095
<b>Input parameter{in}</b>	
<b>high_threshold</b>	analog watchdog high threshold, 0..4095
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### adc\_flag\_get

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

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

<b>Function name</b>	adc_flag_get
<b>Function prototype</b>	FlagStatus adc_flag_get(uint32_t adc_periph , uint32_t adc_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..2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_flag</b>	the adc flag bits
<i>ADC_FLAG_WDE</i>	analog watchdog event flag
<i>ADC_FLAG_EOC</i>	end of group conversion flag
<i>ADC_FLAG_EOIC</i>	end of inserted group conversion flag
<i>ADC_FLAG_STIC</i>	start flag of inserted channel group
<i>ADC_FLAG_STRC</i>	start flag of routine channel group

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

Example:

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

```
FlagStatus flag_value;
```

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

### adc\_flag\_clear

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

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

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

Return value	
-	-

Example:

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

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

### adc\_routine\_software\_startconv\_flag\_get

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

**Table 3-38. Function adc\_routine\_software\_startconv\_flag\_get**

Function name	adc_routine_software_startconv_flag_get
Function prototype	FlagStatus adc_routine_software_startconv_flag_get(uint32_t adc_periph);
Function descriptions	get the bit state of ADCx software routine channel start conversion
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the bit state of ADC0 software routine channel start conversion */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_routine_software_startconv_flag_get(ADC0);
```

### adc\_inserted\_software\_startconv\_flag\_get

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

**Table 3-39. Function adc\_inserted\_software\_startconv\_flag\_get**

Function name	adc_inserted_software_startconv_flag_get
---------------	------------------------------------------

<b>Function prototype</b>	FlagStatus adc_inserted_software_startconv_flag_get(uint32_t adc_periph);
<b>Function descriptions</b>	get the bit state of ADCx software inserted channel start conversion
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the bit state of ADC0 software inserted channel start conversion */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_inserted_software_startconv_flag_get(ADC0);
```

### adc\_interrupt\_flag\_get

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

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

<b>Function name</b>	adc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, uint32_t adc_interrupt);
<b>Function descriptions</b>	get the ADC interrupt bits
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_interrupt</b>	the adc interrupt bits

<i>ADC_INT_WDE</i>	analog watchdog interrupt
<i>ADC_INT_EOC</i>	end of group conversion interrupt
<i>ADC_INT_EOIC</i>	end of inserted group conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
FlagStatus flag_value;
```

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

### adc\_interrupt\_flag\_clear

The description of `adc_interrupt_flag_clear` is shown as below:

**Table 3-41. Function `adc_interrupt_flag_clear`**

<b>Function name</b>	<code>adc_interrupt_flag_clear</code>
<b>Function prototype</b>	<code>void adc_interrupt_flag_clear(uint32_t adc_periph, uint32_t adc_interrupt);</code>
<b>Function descriptions</b>	clear the ADC interrupt bits
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_interrupt</b>	the adc interrupt bits
<i>ADC_INT_WDE</i>	analog watchdog interrupt
<i>ADC_INT_EOC</i>	end of group conversion interrupt
<i>ADC_INT_EOIC</i>	end of inserted group conversion interrupt
<b>Output parameter{out}</b>	

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

Example:

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

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

### adc\_interrupt\_enable

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

**Table 3-42. Function adc\_interrupt\_enable**

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

Example:

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

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

## adc\_interrupt\_disable

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

**Table 3-43. Function adc\_interrupt\_disable**

<b>Function name</b>	adc_interrupt_disable
<b>Function prototype</b>	void adc_interrupt_disable(uint32_t adc_periph , uint32_t adc_interrupt);
<b>Function descriptions</b>	Disable ADC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_periph</b>	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
<b>Input parameter{in}</b>	
<b>adc_interrupt</b>	the adc interrupt
<i>ADC_INT_WDE</i>	analog watchdog interrupt
<i>ADC_INT_EOC</i>	end of group conversion interrupt
<i>ADC_INT_EOIC</i>	end of inserted group conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## adc\_sync\_mode\_config

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

**Table 3-44. 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
ADC_SYNC_MODE_INDEPENDENT	all the ADCs work independently
ADC_DAUL_REGULAR_PARALLEL_INSERTED_PARALLEL	ADC0 and ADC1 work in combined routine parallel & inserted parallel mode
ADC_DAUL_REGULAR_PARALLEL_INSERTED_ROTATION	ADC0 and ADC1 work in combined routine parallel & trigger rotation mode
ADC_DAUL_INSERTED_PARALLEL	ADC0 and ADC1 work in inserted parallel mode
ADC_DAUL_REGULAR_PARALLEL	ADC0 and ADC1 work in routine parallel mode
ADC_DAUL_REGULAR_FOLLOW_UP	ADC0 and ADC1 work in follow-up mode
ADC_DAUL_INSERTED_TRIGGER_ROTATION	ADC0 and ADC1 work in trigger rotation mode
ADC_ALL_REGULAR_PARALLEL_INSERTED_PARALLEL	all ADCs work in combined routine parallel & inserted parallel mode
ADC_ALL_REGULAR_PARALLEL_INSERTED_ROTATION	all ADCs work in combined routine parallel & trigger rotation mode
ADC_ALL_INSERTED_PARALLEL	all ADCs work in inserted parallel mode
ADC_ALL_REGULAR_PARALLEL	all ADCs work in routine parallel mode
ADC_ALL_REGULAR_FOLLOW_UP	all ADCs work in follow-up mode
ADC_ALL_INSERTED_TRIGGER_ROTATION	all ADCs work in trigger rotation mode



<i>N</i>	
<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_DUAL_REGULAR_PARALLEL_INSERTED_PARALLEL);
```

### adc\_sync\_delay\_config

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

**Table 3-45. Function adc\_interrupt\_disable**

<b>Function name</b>	adc_sync_delay_config
<b>Function prototype</b>	void adc_sync_delay_config(uint32_t sample_delay);
<b>Function descriptions</b>	configure the delay between 2 sampling phases in ADC sync modes
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sample_delay</b>	the delay between 2 sampling phases in ADC sync modes
ADC_SYNC_DELAY_x CYCLE(x=5..20)	the delay between 2 sampling phases in ADC sync modes is x ADC clock cycles
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the delay between 2 sampling phases in ADC sync modes */
adc_sync_delay_config (ADC_SYNC_DELAY_5CYCLE);
```

## adc\_sync\_dma\_config

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

**Table 3-46. Function adc\_sync\_dma\_config**

<b>Function name</b>	adc_sync_dma_config
<b>Function prototype</b>	void adc_sync_dma_config(uint32_t dma_mode );
<b>Function descriptions</b>	configure ADC sync DMA mode selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_mode</b>	the adc interrupt
ADC_SYNC_DMA_DISABLE	ADC sync DMA disabled
ADC_SYNC_DMA_MODE0	ADC sync DMA mode 0
ADC_SYNC_DMA_MODE1	ADC sync DMA mode 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC sync DMA mode selection */
adc_sync_dma_config (ADC_SYNC_DMA_MODE0);
```

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

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

**Table 3-47. Function adc\_sync\_dma\_request\_after\_last\_enable**

<b>Function name</b>	adc_sync_dma_request_after_last_enable
<b>Function prototype</b>	void adc_sync_dma_request_after_last_enable(void);
<b>Function descriptions</b>	when SYNCDMA is not equal to 2'b00, the DMA engine issues requests according to the SYNCDMA bits

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

Example:

```
/* when SYNC DMA is not equal to 2'b00, the DMA engine issues requests according to the
SYNC DMA bits */
```

```
adc_sync_dma_request_after_last_enable();
```

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

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

**Table 3-48. Function adc\_sync\_dma\_request\_after\_last\_disable**

<b>Function name</b>	adc_sync_dma_request_after_last_disable
<b>Function prototype</b>	void adc_sync_dma_request_after_last_disable (void);
<b>Function descriptions</b>	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_sync_dma_request_after_last_disable();
```

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

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

**Table 3-49. Function adc\_interrupt\_disable**

<b>Function name</b>	adc_sync_routine_data_read
<b>Function prototype</b>	uint32_t adc_sync_routine_data_read(void);
<b>Function descriptions</b>	read ADC sync routine 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	ADC sync routine data register

Example:

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

```
adc_sync_routine_data_read ();
```

## 3.3. 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 [3.3.1](#), the CAN firmware functions are introduced in chapter [3.3.2](#).

### 3.3.1. Descriptions of Peripheral registers

CAN registers are listed in the table shown as below:

**Table 3-50. CAN Registers**

Registers	Descriptions
CAN_CTL	Control register
CAN_STAT	Status register
CAN_TSTAT	Transmit status register

Registers	Descriptions
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_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

### 3.3.2. Descriptions of Peripheral functions

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

**Table 3-51. CAN firmware function**

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

Function name	Function description
can_working_mode_set	CAN working mode
can_wakeup	CAN wakeup from sleep mode
can_error_get	CAN get error
can_receive_error_number_get	get CAN receive error number
can_transmit_error_number_get	get CAN transmit error number
can_interrupt_enable	CAN interrupt enable
can_interrupt_disable	CAN interrupt disable
can_flag_get	CAN get flag state
can_flag_clear	CAN clear flag state
can_interrupt_flag_get	CAN get interrupt flag state
can_interrupt_flag_clear	CAN clear interrupt flag state

### Structure can\_parameter\_struct

**Table 3-52. Structure can\_parameter\_struct**

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

### Structure can\_transmit\_message\_struct

**Table 3-53. 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[8]	transmit data

## Structure can\_receive\_message\_struct

**Table 3-54. Structure can\_receive\_message\_struct**

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

## Structure can\_filter\_parameter\_struct

**Table 3-55. Structure can\_filter\_parameter\_struct**

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

## can\_deinit

The description of can\_deinit is shown as below:

**Table 3-56. Function can\_deinit**

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

Example:

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

```
can_deinit (CAN0);
```

### can\_struct\_para\_init

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

**Table 3-57. Function can\_struct\_para\_init**

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

Example:

```
can_parameter_struct can_init;
```

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

### can\_init

The description of can\_init is shown as below:

**Table 3-58. 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



CANx(x=0,1)	CAN peripheral selection
<b>Input parameter{in}</b>	
can_parameter_init	CAN parameter initialization stuct, the structure members can refer to members of the structure <a href="#">Table 3-52. Structure can_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
ErrStatus	SUCCESS / ERROR

Example:

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

```
can_init (CAN0);
```

### can\_filter\_init

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

**Table 3-59. Function can\_filter\_init**

<b>Function name</b>	can_filter_init
<b>Function prototype</b>	void can_filter_init(can_filter_parameter_struct* can_filter_parameter_init);
<b>Function descriptions</b>	initialize CAN filter
<b>Precondition</b>	can_struct_para_init()
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
can_filter_parameter_init	CAN filter initialization stuct, the structure members can refer to members of the structure <a href="#">Table 3-55. Structure can_filter_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### can1\_filter\_start\_bank

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

**Table 3-60. 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
1..27	start number
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
can1_filter_start_bank (15);
```

### can\_debug\_freeze\_enable

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

**Table 3-61. 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
CANx(x=0,1)	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 3-62. 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 3-63. 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 3-64. 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 3-65. 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 3-53. Structure can_transmit_message_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	0x00-0x03

Example:

```
/* CAN0 transmit message and return the mailbox number*/
uint8_t transmit_mailbox = 0;
transmit_mailbox = can_message_transmit(CAN0, &transmit_message);
```

## can\_transmit\_states

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

**Table 3-66. 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_enum</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 3-67. Function can\_transmission\_stop**

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

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### can\_message\_receive

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

**Table 3-68. Function can\_message\_receive**

Function name	can_message_receive
Function prototype	void can_message_receive(uint32_t can_periph, uint8_t fifo_number, can_receive_message_struct* receive_message);
Function descriptions	CAN receive message
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
fifo_number	Fifo number
CAN_FIFOx	CAN_FIFOx(x=0,1)
Input parameter{in}	
receive_message	CAN message receive struct, the structure members can refer to members of the structure <a href="#">Table 3-54. Structure can_receive_message_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CAN0 FIFO0 receive message */
can_message_receive(CAN0, CAN_FIFO0, &receive_message);
```

### can\_fifo\_release

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

**Table 3-69. Function can\_fifo\_release**

Function name	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 3-70. 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 3-71. 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 3-72. 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 3-73. Function can\_error\_get**

<b>Function name</b>	can_error_get
<b>Function prototype</b>	can_error_enum can_error_get(uint32_t can_periph);
<b>Function descriptions</b>	get CAN error type
<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>can_error_enum</b>	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 3-74. Function can\_receive\_error\_number\_get**

<b>Function name</b>	can_receive_error_number_get
<b>Function prototype</b>	uint8_t can_receive_error_number_get(uint32_t can_periph);
<b>Function descriptions</b>	get CAN receive error number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral

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

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

**Table 3-75. Function can\_transmit\_error\_number\_get**

Function name	can_transmit_error_number_get
Function prototype	uint8_t can_transmit_error_number_get(uint32_t can_periph);
Function descriptions	get CAN transmit error number
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
uint8_t	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 3-76. Function can\_flag\_get**

Function name	can_flag_get
Function prototype	FlagStatus can_flag_get(uint32_t can_periph, can_flag_enum flag);
Function descriptions	get CAN flag state
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral

CANx(x=0,1)	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>flag</b>	CAN flags
CAN_FLAG_MTE2	mailbox 2 transmit error
CAN_FLAG_MTE1	mailbox 1 transmit error
CAN_FLAG_MTE0	mailbox 0 transmit error
CAN_FLAG_MTF2	mailbox 2 transmit finished
CAN_FLAG_MTF1	mailbox 1 transmit finished
CAN_FLAG_MTF0	mailbox 0 transmit finished
CAN_FLAG_RFO0	receive FIFO0 overfull
CAN_FLAG_RFF0	receive FIFO0 full
CAN_FLAG_RFO1	receive FIFO1 overfull
CAN_FLAG_RFF1	receive FIFO1 full
CAN_FLAG_BOERR	bus-off error
CAN_FLAG_PERR	passive error
CAN_FLAG_WERR	warning error
<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 3-77. 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
CANx(x=0,1)	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>flag</b>	CAN flags
CAN_FLAG_MTE2	mailbox 2 transmit error
CAN_FLAG_MTE1	mailbox 1 transmit error
CAN_FLAG_MTE0	mailbox 0 transmit error

<i>CAN_FLAG_MTF2</i>	mailbox 2 transmit finished
<i>CAN_FLAG_MTF1</i>	mailbox 1 transmit finished
<i>CAN_FLAG_MTF0</i>	mailbox 0 transmit finished
<i>CAN_FLAG_RFO0</i>	receive FIFO0 overfull
<i>CAN_FLAG_RFF0</i>	receive FIFO0 full
<i>CAN_FLAG_RFO1</i>	receive FIFO1 overfull
<i>CAN_FLAG_RFF1</i>	receive FIFO1 full
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### can\_interrupt\_enable

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

**Table 3-78. 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
<i>CANx(x=0,1)</i>	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	Interrupt type
<i>CAN_INT_TME</i>	transmit mailbox empty interrupt enable
<i>CAN_INT_RFNE0</i>	receive FIFO0 not empty interrupt enable
<i>CAN_INT_RFF0</i>	receive FIFO0 full interrupt enable
<i>CAN_INT_RFO0</i>	receive FIFO0 overfull interrupt enable
<i>CAN_INT_RFNE1</i>	receive FIFO1 not empty interrupt enable
<i>CAN_INT_RFF1</i>	receive FIFO1 full interrupt enable
<i>CAN_INT_RFO1</i>	receive FIFO1 overfull interrupt enable
<i>CAN_INT_WERR</i>	warning error interrupt enable
<i>CAN_INT_PERR</i>	passive error interrupt enable
<i>CAN_INT_BO</i>	bus-off interrupt enable
<i>CAN_INT_ERRN</i>	error number interrupt enable
<i>CAN_INT_ERR</i>	error interrupt enable

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

Example:

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

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

### can\_interrupt\_disable

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

**Table 3-79. Function can\_interrupt\_disable**

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

-	-
---	---

Example:

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

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

### can\_interrupt\_flag\_get

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

**Table 3-80. Function can\_interrupt\_flag\_get**

<b>Function name</b>	can_interrupt_flag_get
<b>Function prototype</b>	FlagStatus can_interrupt_flag_get(uint32_t can_periph, can_interrupt_flag_enum flag);
<b>Function descriptions</b>	get CAN interrupt flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>can_periph</b>	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
<b>Input parameter{in}</b>	
<b>flag</b>	CAN interrupt flags
CAN_INT_FLAG_SLPIF	status change interrupt flag of sleep working mode entering
CAN_INT_FLAG_WUIF	status change interrupt flag of wakeup from sleep working mode
CAN_INT_FLAG_ERRIF	error interrupt flag
CAN_INT_FLAG_MTF2	mailbox 2 transmit finished interrupt flag
CAN_INT_FLAG_MTF1	mailbox 1 transmit finished interrupt flag
CAN_INT_FLAG_MTF0	mailbox 0 transmit finished interrupt flag
CAN_INT_FLAG_RFO0	receive FIFO0 overfull interrupt flag
CAN_INT_FLAG_RFF0	receive FIFO0 full interrupt flag
CAN_INT_FLAG_RFO1	receive FIFO1 overfull interrupt flag
CAN_INT_FLAG_RFF1	receive FIFO1 full interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<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 3-81. Function can\_interrupt\_flag\_clear**

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

Example:

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

## 3.4. CRC

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

### 3.4.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

**Table 3-82. CRC Registers**

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

### 3.4.2. Descriptions of Peripheral functions

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

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

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

#### crc\_deinit

The description of crc\_deinit is shown as below:

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

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

Example:

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



## crc\_data\_register\_reset

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

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

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

Example:

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

## crc\_data\_register\_read

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

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

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

Example:

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

## crc\_free\_data\_register\_read

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

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

<b>Function name</b>	crc_free_data_register_read
<b>Function prototype</b>	uint8_t crc_free_data_register_read(void);
<b>Function descriptions</b>	read the value of the free data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	8-bit value of the free data register (0-0xFF)

Example:

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

## crc\_free\_data\_register\_write

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

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

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

Example:

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

## crc\_single\_data\_calculate

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

**Table 3-89. Function crc\_single\_data\_calculate**

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

Example:

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

## crc\_block\_data\_calculate

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

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

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

Example:

```

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

#define BUFFER_SIZE    6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

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

rcu_periph_clock_enable(RCU_CRC);

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

```

## 3.5. CTC

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

### 3.5.1. Descriptions of Peripheral registers

CTC registers are listed in the table shown as below:

**Table 3-91. CTC Registers**

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

### 3.5.2. Descriptions of Peripheral functions

CTC registers are listed in the table shown as below:

**Table 3-92. 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

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

## ctc\_deinit

The description of ctc\_deinit is shown as below:

**Table 3-93. Function ctc\_deinit**

Function name	ctc_deinit
Function prototype	void ctc_deinit (void)
Function descriptions	Reset CTC peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable

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

Example:

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

```
ctc_deinit ();
```

### ctc\_counter\_enable

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

**Table 3-94. Function ctc\_counter\_enable**

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

Example:

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

```
ctc_counter_enable ();
```

### ctc\_counter\_disable

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

Table 3-95. Function `ctc_counter_disable`

Function name	<code>ctc_counter_disable</code>
Function prototype	<code>void ctc_counter_disable (void);</code>
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 3-96. Function `ctc_irc48m_trim_value_config`

Function name	<code>ctc_irc48m_trim_value_config</code>
Function prototype	<code>void ctc_irc48m_trim_value_config(uint8_t trim_value);</code>
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 3-97. Function ctc\_software\_refsource\_pulse\_generate**

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

Example:

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

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

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

**Table 3-98. Function ctc\_hardware\_trim\_mode\_config**

<b>Function name</b>	ctc_hardware_trim_mode_config
<b>Function prototype</b>	void ctc_hardware_trim_mode_config(uint32_t hardmode);
<b>Function descriptions</b>	configure hardware automatically trim mode
<b>Precondition</b>	-
<b>The called functions</b>	-



Input parameter{in}	
<b>hardmode</b>	hardware automatically trim mode enable or disable
<i>CTC_HARDWARE_TRIM_MODE_ENABLE</i>	hardware automatically trim mode enable
<i>CTC_HARDWARE_TRIM_MODE_DISABLE</i>	hardware automatically trim mode disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
ctc_hardware_trim_mode_config (CTC_HARDWARE_TRIM_MODE_ENABLE);
```

### ctc\_refsource\_polarity\_config

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

**Table 3-99. Function ctc\_refsource\_polarity\_config**

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

-	-
---	---

Example:

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

```
ctc_refsource_polarity_config (CTC_REFSOURCE_POLARITY_RISING);
```

### ctc\_refsource\_signal\_select

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

**Table 3-100. Function ctc\_refsource\_signal\_select**

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

Example:

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

```
ctc_refsource_signal_select (CTC_REFSOURCE_LXTAL);
```

### ctc\_refsource\_prescaler\_config

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

**Table 3-101. Function ctc\_refsource\_prescaler\_config**

<b>Function name</b>	ctc_refsource_prescaler_config
----------------------	--------------------------------

<b>Function prototype</b>	void ctc_refsource_prescaler_config(uint32_t prescaler);
<b>Function descriptions</b>	configure reference signal source prescaler
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>prescaler</b>	Prescaler factor
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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
ctc_refsource_prescaler_config (CTC_REFSOURCE_PSC_DIV2);
```

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

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

Table 3-102. 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 3-103. 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 3-104. Function ctc\_counter\_capture\_value\_read**

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

Example:

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

### ctc\_counter\_direction\_read

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

**Table 3-105. Function ctc\_counter\_direction\_read**

<b>Function name</b>	ctc_counter_direction_read
<b>Function prototype</b>	FlagStatus ctc_counter_direction_read(void);
<b>Function descriptions</b>	read CTC trim counter direction
<b>Precondition</b>	-

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

Example:

```
/* read ctc counter direction */
```

```
FlagStatus ctc_direction = SET;
```

```
ctc_direction = ctc_counter_direction_read ();
```

### ctc\_counter\_reload\_value\_read

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

**Table 3-106. Function ctc\_counter\_reload\_value\_read**

Function name	ctc_counter_reload_value_read
Function prototype	uint16_t ctc_counter_reload_value_read(void);
Function descriptions	read CTC counter reload value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	Read 16-bit data of counter reload value (0x0000 - 0xFFFF)

Example:

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

```
uint16_t ctc_reload_value = 0;
```

```
ctc_reload_value = ctc_counter_reload_value_read ();
```

### ctc\_irc48m\_trim\_value\_read

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

**Table 3-107. Function ctc\_irc48m\_trim\_value\_read**

<b>Function name</b>	ctc_irc48m_trim_value_read
<b>Function prototype</b>	uint8_t ctc_irc48m_trim_value_read(void);
<b>Function descriptions</b>	read the IRC48M trim value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint8_t	the 8-bit IRC48M trim value (0-63)

Example:

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

### ctc\_interrupt\_enable

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

**Table 3-108. Function ctc\_interrupt\_enable**

<b>Function name</b>	ctc_interrupt_enable
<b>Function prototype</b>	void ctc_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable the CTC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
interrupt	CTC interrupt
CTC_INT_CKOK	clock trim OK interrupt

<i>CTC_INT_CKWARN</i>	clock trim warning interrupt
<i>CTC_INT_ERR</i>	error interrupt
<i>CTC_INT_EREf</i>	expect reference interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* 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 3-109. Function ctc\_interrupt\_disable**

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



Example:

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

### ctc\_interrupt\_flag\_get

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

**Table 3-110. 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_EREFP	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	
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 3-111. Function ctc\_interrupt\_flag\_clear**

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

Example:

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

```
ctc_interrupt_flag_clear (CTC_INT_FLAG_CKOK);
```

## ctc\_flag\_get

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

**Table 3-112. Function ctc\_flag\_get**

<b>Function name</b>	ctc_flag_get
<b>Function prototype</b>	FlagStatus ctc_flag_get(uint32_t flag);
<b>Function descriptions</b>	get CTC status flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	CTC status flag
CTC_FLAG_CKOK	clock trim OK interrupt flag
CTC_FLAG_CKWARN	clock trim warning interrupt flag
CTC_FLAG_ERR	error interrupt flag
CTC_FLAG_EREFP	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

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

## ctc\_flag\_clear

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

**Table 3-113. Function ctc\_flag\_clear**

<b>Function name</b>	ctc_flag_clear
<b>Function prototype</b>	void ctc_flag_clear (uint32_t flag);

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

Example:

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

```
ctc_flag_clear (CTC_FLAG_CKOK);
```

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

### 3.6.1. Descriptions of Peripheral registers

DAC registers are listed in the table shown as below:

**Table 3-114. DAC Registers**

Registers	Descriptions
DAC_CTL0	DACx control register 0
DAC_SWT	DACx software trigger register

Registers	Descriptions
DAC_OUT0_R12DH	DACx_OUT0 12-bit right-aligned data holding register
DAC_OUT0_L12DH	DACx_OUT0 12-bit left-aligned data holding register
DAC_OUT0_R8DH	DACx_OUT0 8-bit right-aligned data holding register
DAC_OUT1_R12DH	DACx_OUT1 12-bit right-aligned data holding register
DAC_OUT1_L12DH	DACx_OUT1 12-bit left-aligned data holding register
DAC_OUT1_R8DH	DACx_OUT1 8-bit right-aligned data holding register
DACC_R12DH	DACx concurrent mode 12-bit right-aligned data holding register
DACC_L12DH	DACx concurrent mode 12-bit left-aligned data holding register
DACC_R8DH	DACx concurrent mode 8-bit right-aligned data holding register
DAC_OUT0_DO	DACx_OUT0 data output register
DAC_OUT1_DO	DACx_OUT1 data output register
DAC_STAT0	DACx status register 0

### 3.6.2. Descriptions of Peripheral functions

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

**Table 3-115. 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
dac_output_buffer_enable	enable DAC output buffer
dac_output_buffer_disable	disable DAC output buffer
dac_output_value_get	get DAC output value
dac_data_set	set DAC data holding register value
dac_trigger_enable	enable DAC trigger
dac_trigger_disable	disable DAC trigger
dac_trigger_source_config	configure DAC trigger source
dac_software_trigger_enable	enable DAC software trigger
dac_wave_mode_config	configure DAC wave mode
dac_lfsr_noise_config	configure DAC LFSR noise mode
dac_triangle_noise_config	configure DAC triangle noise mode
dac_concurrent_enable	enable DAC concurrent mode
dac_concurrent_disable	disable DAC concurrent mode
dac_concurrent_software_trigger_enable	enable DAC concurrent software trigger
dac_concurrent_output_buffer_enable	enable DAC concurrent buffer function
dac_concurrent_output_buffer_disable	disable DAC concurrent buffer function
dac_concurrent_data_set	set DAC concurrent mode data holding register value
dac_flag_get	get DAC flag

Function name	Function description
<code>dac_flag_clear</code>	clear 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 DAC interrupt flag
<code>dac_interrupt_flag_clear</code>	clear DAC interrupt flag

## **dac\_deinit**

The description of `dac_deinit` is shown as below:

**Table 3-116. 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>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
dac_deinit(DAC0);
```

## **dac\_enable**

The description of `dac_enable` is shown as below:

**Table 3-117. Function `dac_enable`**

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

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 3-118. Function dac\_disable**

Function name	dac_disable
Function prototype	void dac_disable(uint32_t dac_periph, uint8_t dac_out);
Function descriptions	disable DAC
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 = 0,1)
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 3-119. Function dac\_dma\_enable**

Function name	dac_dma_enable
Function prototype	void dac_dma_enable(uint32_t dac_periph, uint8_t dac_out);
Function descriptions	enable DAC DMA function
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 = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

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 3-120. Function dac\_dma\_disable**

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

Example:

```
/* disable DAC0_OUT0 DMA function */
dac_dma_disable(DAC0, DAC_OUT0);
```

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

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



Table 3-121. 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>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b>dac_out</b>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

The description of `dac_output_buffer_disable` is shown as below:

Table 3-122. 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>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b>dac_out</b>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DAC0_OUT0 output buffer */
```

```
dac_output_buffer_disable(DAC0, DAC_OUT0);
```

## dac\_output\_value\_get

The description of `dac_output_value_get` is shown as below:

**Table 3-123. 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, uint8_t dac_out);</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 = 0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	DAC output data (0~4095)

Example:

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

```
uint16_t data=0;
```

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

## dac\_data\_set

The description of `dac_data_set` is shown as below:

**Table 3-124. Function `dac_data_set`**

<b>Function name</b>	<code>dac_data_set</code>
<b>Function prototype</b>	<code>void dac_data_set(uint32_t dac_periph, uint8_t dac_out, uint32_t dac_align, uint16_t data);</code>
<b>Function descriptions</b>	set DAC data holding register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b>dac_out</b>	DAC output

<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)
<b>Input parameter{in}</b>	
<b>dac_align</b>	DAC data alignment mode
<i>DAC_ALIGN_12B_R</i>	12-bit right-aligned data
<i>DAC_ALIGN_12B_L</i>	12-bit left-aligned data
<i>DAC_ALIGN_8B_R</i>	8-bit right-aligned data
<b>Input parameter{in}</b>	
<b>data</b>	data to be loaded (0~4095)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

**Table 3-125. Function dac\_trigger\_enable**

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

Example:

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

## dac\_trigger\_disable

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

**Table 3-126. Function dac\_trigger\_disable**

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

Example:

```
/* disable DAC0_OUT0 trigger */
dac_trigger_disable(DAC0, DAC_OUT0);
```

## dac\_trigger\_source\_config

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

**Table 3-127. 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 = 0,1)
<b>Input parameter{in}</b>	
<b>triggersource</b>	external trigger of DAC

<i>DAC_TRIGGER_T5_TRGO</i> O	TIMER5 TRGO
<i>DAC_TRIGGER_T7_TRGO</i> O	TIMER7 TRGO
<i>DAC_TRIGGER_T6_TRGO</i> O	TIMER6 TRGO
<i>DAC_TRIGGER_T4_TRGO</i> O	TIMER4 TRGO
<i>DAC_TRIGGER_T1_TRGO</i> O	TIMER1 TRGO
<i>DAC_TRIGGER_T3_TRGO</i> O	TIMER3 TRGO
<i>DAC_TRIGGER_EXTI_9</i>	EXTI interrupt line9 event
<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_software_trigger_enable` is shown as below:

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

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

-	-
---	---

Example:

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

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

The description of dac\_wave\_mode\_config is shown as below:

**Table 3-129. Function dac\_wave\_mode\_config**

Function name	dac_wave_mode_config
Function prototype	void dac_wave_mode_config(uint32_t dac_periph, uint8_t dac_out, uint32_t wave_mode);
Function descriptions	configure DAC wave mode
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 = 0,1)
Input parameter{in}	
<b>wave_mode</b>	DAC wave mode
<i>DAC_WAVE_DISABLE</i>	wave mode disable
<i>DAC_WAVE_MODE_LFSR</i>	LFSR noise mode
<i>DAC_WAVE_MODE_TRIANGLE</i>	triangle noise mode
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 3-130. Function dac\_lfsr\_noise\_config**

Function name	dac_lfsr_noise_config
---------------	-----------------------

<b>Function prototype</b>	void dac_lfsr_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t unmask_bits);
<b>Function descriptions</b>	configure DAC LFSR noise mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
<b>Input parameter{in}</b>	
<b>dac_out</b>	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
<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 3-131. Function dac\_triangle\_noise\_config**

<b>Function name</b>	dac_triangle_noise_config
<b>Function prototype</b>	void dac_triangle_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t amplitude);
<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 = 0,1)
<b>Input parameter{in}</b>	
<b>amplitude</b>	the amplitude of the triangle
<i>DAC_TRIANGLE_AMPLIT</i>	$x = 2^n - 1$ (n = 1..12)

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

Example:

```
/* configure DAC0_OUT0 triangle noise mode */
dac_triangle_noise_config(DAC0, DAC_OUT0, DAC_TRIANGLE_AMPLITUDE_1);
```

### **dac\_concurrent\_enable**

The description of `dac_concurrent_enable` is shown as below:

**Table 3-132. Function `dac_concurrent_enable`**

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

Example:

```
/* enable DAC0 concurrent mode */
dac_concurrent_enable(DAC0);
```

### **dac\_concurrent\_disable**

The description of `dac_concurrent_disable` is shown as below:

**Table 3-133. Function `dac_concurrent_disable`**

<b>Function name</b>	<code>dac_concurrent_disable</code>
<b>Function prototype</b>	<code>void dac_concurrent_disable(uint32_t dac_periph);</code>
<b>Function descriptions</b>	disable DAC concurrent mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>dac_periph</b>	DAC peripheral



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

Example:

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

```
dac_concurrent_disable(DAC0);
```

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

The description of `dac_concurrent_software_trigger_enable` is shown as below:

**Table 3-134. Function `dac_concurrent_software_trigger_enable`**

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

Example:

```
/* enable DAC0 concurrent software trigger */
```

```
dac_concurrent_software_trigger_enable(DAC0);
```

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

The description of `dac_concurrent_output_buffer_enable` is shown as below:

**Table 3-135. Function `dac_concurrent_output_buffer_enable`**

<b>Function name</b>	<code>dac_concurrent_output_buffer_enable</code>
<b>Function prototype</b>	<code>void dac_concurrent_output_buffer_enable(uint32_t dac_periph);</code>
<b>Function descriptions</b>	enable DAC concurrent buffer function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dac_periph</b>	DAC peripheral

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

Example:

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

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

The description of `dac_concurrent_output_buffer_disable` is shown as below:

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

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

Example:

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

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

The description of `dac_concurrent_data_set` is shown as below:

**Table 3-137. Function `dac_concurrent_data_set`**

<b>Function name</b>	<code>dac_concurrent_data_set</code>
<b>Function prototype</b>	<code>void dac_concurrent_data_set(uint32_t dac_periph, uint32_t dac_align, uint16_t data0, uint16_t data1);</code>
<b>Function descriptions</b>	set DAC concurrent mode data holding register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>dac_periph</b>	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
<b>Input parameter{in}</b>	
<b>dac_align</b>	DAC data alignment mode
<i>DAC_ALIGN_8B_R</i>	8-bit right-aligned data
<i>DAC_ALIGN_12B_R</i>	12-bit right-aligned data
<i>DAC_ALIGN_12B_L</i>	12-bit left-aligned data
<b>Input parameter{in}</b>	
<b>data0</b>	DACx_OUT0 data to be loaded (0~4095)
<b>Input parameter{in}</b>	
<b>data1</b>	DACx_OUT1 data to be loaded (0~4095)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

The description of `dac_flag_get` is shown as below:

**Table 3-138. 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)
<b>Input parameter{in}</b>	
<b>flag</b>	the DAC status flags
<i>DAC_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun flag
<i>DAC_FLAG_DDUDR1</i>	DACx_OUT1 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 3-139. Function dac\_flag\_clear**

<b>Function name</b>	dac_flag_clear
<b>Function prototype</b>	void dac_flag_clear(uint32_t dac_periph, uint32_t flag);
<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)
<b>Input parameter{in}</b>	
<b>flag</b>	DAC flag
<i>DAC_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun flag
<i>DAC_FLAG_DDUDR1</i>	DACx_OUT1 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 3-140. 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)
<b>Input parameter{in}</b>	
<b>interrupt</b>	the DAC interrupt

<i>DAC_INT_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt
<i>DAC_INT_DDUDR1</i>	DACx_OUT1 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 3-141. Function `dac_interrupt_disable`**

<b>Function name</b>	<code>dac_interrupt_disable</code>
<b>Function prototype</b>	<code>void dac_interrupt_disable(uint32_t dac_periph, uint32_t interrupt);</code>
<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)
<b>Input parameter{in}</b>	
<b>interrupt</b>	the DAC interrupt
<i>DAC_INT_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt
<i>DAC_INT_DDUDR1</i>	DACx_OUT1 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 3-142. Function `dac_interrupt_flag_get`**

<b>Function name</b>	<code>dac_interrupt_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus dac_interrupt_flag_get(uint32_t dac_periph, uint32_t</code>

	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)
<b>Input parameter{in}</b>	
<b>int_flag</b>	DAC interrupt flag
<i>DAC_INT_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt flag
<i>DAC_INT_FLAG_DDUDR1</i>	DACx_OUT1 DMA underrun interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<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 3-143. 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)
<b>Input parameter{in}</b>	
<b>int_flag</b>	DAC interrupt flag
<i>DAC_INT_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt flag
<i>DAC_INT_FLAG_DDUDR1</i>	DACx_OUT1 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);
```

## 3.7. DBG

The DBG hold unit helps debugger to debug power saving mode. The DBG registers are listed in chapter [3.7.1](#). the DBG firmware functions are introduced in chapter [3.7.2](#).

### 3.7.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

**Table 3-144. DBG Registers**

Registers	Descriptions
DBG_ID	DBG ID code register
DBG_CTL	DBG control register

### 3.7.2. Descriptions of Peripheral functions

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

**Table 3-145. DBG firmware function**

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

#### Enum dbg\_periph\_enum

**Table 3-146. Enum dbg\_periph\_enum**

Member name	Function description
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted

DBG_TIMER0_HOLD	hold TIMER0 counter when core is halted
DBG_TIMER1_HOLD	hold TIMER1 counter when core is halted
DBG_TIMER2_HOLD	hold TIMER2 counter when core is halted
DBG_TIMER3_HOLD	hold TIMER3 counter when core is halted
DBG_CAN0_HOLD	debug CAN0 kept when core is halted
DBG_I2C0_HOLD	hold I2C0 smbus when core is halted
DBG_I2C1_HOLD	hold I2C1 smbus when core is halted
DBG_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_TIMER7_HOLD	hold TIMER7 counter when core is halted
DBG_CAN1_HOLD	debug CAN1 kept when core is halted
DBG_TIMER11_HOLD	hold TIMER11 counter when core is halted
DBG_TIMER12_HOLD	hold TIMER12 counter when core is halted
DBG_TIMER13_HOLD	hold TIMER13 counter when core is halted
DBG_TIMER8_HOLD	hold TIMER8 counter when core is halted
DBG_TIMER9_HOLD	hold TIMER9 counter when core is halted
DBG_TIMER10_HOLD	hold TIMER10 counter when core is halted

## dbg\_id\_get

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

**Table 3-147. 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>	
-	-



Output parameter{out}	
-	-
Return value	
uint32_t	DBG_ID code (0-0xFFFFFFFF)

Example:

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

```
uint32_t id_value = 0;
```

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

### dbg\_low\_power\_enable

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

**Table 3-148. Function dbg\_low\_power\_enable**

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

Example:

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

```
dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

### dbg\_low\_power\_disable

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

**Table 3-149. Function dbg\_low\_power\_disable**

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

Example:

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

```
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

### dbg\_periph\_enable

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

**Table 3-150. Function dbg\_periph\_enable**

<b>Function name</b>	dbg_periph_enable
<b>Function prototype</b>	void dbg_periph_enable(dbg_periph_enum dbg_periph);

<b>Function descriptions</b>	Enable peripheral behavior when the mcu is in debug mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dbg_periph</b>	Peripheral refer to <a href="#">Table 3-146. Enum dbg_periph_enum</a>
<i>DBG_FWDGT_HOLD</i>	debug FWDGT kept when core is halted
<i>DBG_WWDGT_HOLD</i>	debug WWDGT kept when core is halted
<i>DBG_CANx_HOLD</i>	x=0,1, hold CANx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1, hold I2Cx smbus when core is halted
<i>DBG_TIMERx_HOLD</i>	x=0,1,2,3,4,5,6,7,8,9,10,11,12,13, hold TIMERx counter when core is halted
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
dbg_periph_enable(DBG_TIMER0_HOLD);
```

### dbg\_periph\_disable

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

**Table 3-151. Function dbg\_periph\_disable**

<b>Function name</b>	dbg_periph_disable
<b>Function prototype</b>	void dbg_periph_disable(dbg_periph_enum dbg_periph);
<b>Function descriptions</b>	Disable peripheral behavior when the mcu is in debug mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dbg_periph</b>	peripheral refer to <a href="#">Table 3-146. Enum dbg_periph_enum</a>
<i>DBG_FWDGT_HOLD</i>	debug FWDGT kept when core is halted

<i>DBG_WWDGT_HOLD</i>	debug WWDGT kept when core is halted
<i>DBG_CANx_HOLD</i>	x=0,1, hold CANx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1, hold I2Cx smbus when core is halted
<i>DBG_TIMERx_HOLD</i>	x=0,1,2,3,4,5,6,7,8,9,10,11,12,13, hold TIMERx counter when core is halted
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
dbg_periph_disable(DBG_TIMER0_HOLD);
```

### dbg\_trace\_pin\_enable

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

**Table 3-152. 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 3-153. Function dbg\_trace\_pin\_disable**

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

Example:

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

## 3.8. DCI

The DCI is a parallel interface able to capture video or picture from a camera via Digital Camera Interface. The DCI registers are listed in chapter [3.8.1](#). the DCI firmware functions are introduced in chapter [3.8.2](#).

### 3.8.1. Descriptions of Peripheral registers

DCI registers are listed in the table shown as below:

**Table 3-154. DCI Registers**

Registers	Descriptions
DCI_CTL	DCI control register
DCI_STAT0	DCI status register 0
DCI_STAT1	DCI status register 1

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

### 3.8.2. Descriptions of Peripheral functions

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

**Table 3-155. DCI firmware function**

Function name	Function description
dci_deinit	DCI deinit
dci_init	initialize DCI registers
dci_enable	enable DCI function
dci_disable	disble DCI function
dci_capture_enable	enable DCI capture
dci_capture_disable	disble DCI capture
dci_jpeg_enable	enable DCI jpeg mode
dci_jpeg_disable	disble DCI jpeg mode
dci_crop_window_enable	enable cropping window function
dci_crop_window_disable	disble cropping window function
dci_crop_window_config	config DCI cropping window
dci_embedded_sync_enable	enable embedded synchronous mode
dci_embedded_sync_disable	disble embedded synchronous mode
dci_sync_codes_config	config synchronous codes in embedded synchronous mode
dci_sync_codes_unmask_config	config synchronous codes unmask in embedded synchronous

Function name	Function description
	mode
dc_i_data_read	read DCI data register
dc_i_flag_get	get specified flag
dc_i_interrupt_enable	enable specified DCI interrupt
dc_i_interrupt_disable	disable specified DCI interrupt
dc_i_interrupt_flag_get	get specified interrupt flag
dc_i_interrupt_flag_clear	clear specified interrupt flag

### Structure dc\_i\_parameter\_struct

**Table 3-156. Structure dc\_i\_parameter\_struct**

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

### dc\_i\_deinit

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

**Table 3-157. Function dc\_i\_deinit**

Function name	dc_i_deinit
Function prototype	void dc_i_deinit(void);

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

Example:

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

```
dc_i_deinit ( );
```

### dc\_i\_init

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

**Table 3-158. Function dc\_i\_init**

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

Example:

```
/* initialize DCI registers */
```

```
dc_i_parameter_struct dc_i_struct;
```



```

dci_struct.capture_mode = DCI_CAPTURE_MODE_CONTINUOUS;

dci_struct.clock_polarity = DCI_CK_POLARITY_RISING;

dci_struct.hsync_polarity = DCI_HSYNC_POLARITY_LOW;

dci_struct.vsync_polarity = DCI_VSYNC_POLARITY_LOW;

dci_struct.frame_rate = DCI_FRAME_RATE_ALL;

dci_struct.interface_format = DCI_INTERFACE_FORMAT_8BITS;

dci_init (&dci_struct);

```

## dci\_enable

The description of dci\_enable is shown as below:

**Table 3-159. Function dci\_enable**

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

Example:

```

/* enable DCI function */

dci_enable( );

```

## dci\_disable

The description of dci\_disable is shown as below:

**Table 3-160. Function dci\_disable**

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

<b>Function descriptions</b>	disable DCI function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DCI function */
```

```
dc_i_disable( );
```

### dc\_i\_capture\_enable

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

**Table 3-161. Function dc\_i\_capture\_enable**

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

Example:

```
/* enable DCI capture */
```

```
dc_i_capture_enable( );
```

## dc\_i\_capture\_disable

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

**Table 3-162. Function dc\_i\_capture\_disable**

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

Example:

```
/* disable DCI capture */
dc_i_capture_disable( );
```

## dc\_i\_peg\_enable

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

**Table 3-163. Function dc\_i\_peg\_enable**

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

Return value	
-	-

Example:

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

```
dci_jpeg_enable( );
```

### dci\_jpeg\_disable

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

**Table 3-164. Function dci\_jpeg\_disable**

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

Example:

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

```
dci_jpeg_disable( );
```

### dci\_crop\_window\_enable

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

**Table 3-165. Function dci\_crop\_window\_enable**

Function name	dci_crop_window_enable
Function prototype	void dci_crop_window_enable(void);
Function descriptions	enable cropping window function

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

Example:

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

```
dc_i_crop_window_enable( );
```

### dc\_i\_crop\_window\_disable

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

**Table 3-166. Function dc\_i\_crop\_window\_disable**

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

Example:

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

```
dc_i_crop_window_disable( );
```

## dc\_i\_crop\_window\_config

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

**Table 3-167. Function dc\_i\_crop\_window\_config**

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

Example:

```
/* config DCI cropping window */
dc_i_crop_window_config (0x800, 0x600, 0x100, 0x100);
```

## dc\_i\_embedded\_sync\_enable

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

**Table 3-168. Function dc\_i\_embedded\_sync\_enable**

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

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

Example:

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

```
dc_i_embedded_sync_enable( );
```

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

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

**Table 3-169. Function dc\_i\_embedded\_sync\_disable**

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

Example:

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

```
dc_i_embedded_sync_disable( );
```

## dc\_i\_sync\_codes\_config

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

**Table 3-170. Function dc\_i\_sync\_codes\_config**

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

Example:

```
/* config synchronous codes in embedded synchronous mode */
dc_i_sync_codes_config (0x10, 0x10, 0x20, 0x20);
```

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

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

**Table 3-171. Function dc\_i\_sync\_codes\_unmask\_config**

<b>Function name</b>	dc_i_sync_codes_unmask_config
<b>Function prototype</b>	void dc_i_sync_codes_unmask_config(uint8_t frame_start, uint8_t line_start,



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

Example:

```
/* config synchronous codes unmask in embedded synchronous mode */
dci_sync_codes_unmask_config (0x10, 0x10, 0x20, 0x20);
```

### dc\_i\_data\_read

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

**Table 3-172. Function dc\_i\_data\_read**

<b>Function name</b>	dc_i_data_read
<b>Function prototype</b>	uint32_t dc_i_data_read(void);
<b>Function descriptions</b>	read DCI data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-

Output parameter{out}	
-	-
Return value	
uint32_t	0x00 – 0xFFFFFFFF

Example:

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

### dc\_i\_flag\_get

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

**Table 3-173. Function dc\_i\_flag\_get**

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

FlagStatus	FlagStatus: SET or RESET
------------	--------------------------

Example:

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

```
FlagStatus status = dci_flag_get (DCI_FLAG_HS);
```

### dc\_i\_interrupt\_enable

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

**Table 3-174. Function dc\_i\_interrupt\_enable**

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

Example:

```
/* enable specified DCI interrupt */
```

```
dc_i_interrupt_enable (DCI_INT_EF);
```

### dc\_i\_interrupt\_disable

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

Table 3-175. Function dci\_interrupt\_disable

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

Example:

```
/* disable specified DCI interrupt */
dci_interrupt_disable (DCI_INT_EF);
```

### dci\_interrupt\_flag\_get

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

Table 3-176. Function dci\_interrupt\_flag\_get

Function name	dci_interrupt_flag_get
Function prototype	FlagStatus dci_interrupt_flag_get(uint32_t interrupt);
Function descriptions	get specified interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	

<b>interrupt</b>	DCI interrupt
<i>DCI_INT_FLAG_EF</i>	end of frame interrupt flag
<i>DCI_INT_FLAG_OVR</i>	FIFO overrun interrupt flag
<i>DCI_INT_FLAG_ESE</i>	embedded synchronous error interrupt flag
<i>DCI_INT_FLAG_VSYN</i> <i>C</i>	vsync interrupt flag
<i>DCI_INT_FLAG_EL</i>	end of line interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
FlagStatus	FlagStatus: SET or RESET

Example:

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

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

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

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

**Table 3-177. Function dc\_i\_interrupt\_flag\_clear**

<b>Function name</b>	dc_i_interrupt_flag_clear
<b>Function prototype</b>	void dc_i_interrupt_flag_clear(uint32_t interrupt);
<b>Function descriptions</b>	clear specified interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	DCI interrupt
<i>DCI_INT_FLAG_EF</i>	end of frame interrupt flag
<i>DCI_INT_FLAG_OVR</i>	FIFO overrun interrupt flag
<i>DCI_INT_FLAG_ESE</i>	embedded synchronous error interrupt flag
<i>DCI_INT_FLAG_VSYN</i> <i>C</i>	vsync interrupt flag

<i>DCI_INT_FLAG_EL</i>	end of line interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*clear specified interrupt flag */
```

```
dci_interrupt_flag_clear (DCI_INT_FLAG_EF);
```

## 3.9. DMA

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

### 3.9.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

**Table 3-178. DMA Registers**

Registers	Descriptions
DMA_INTF	Interrupt flag register
DMA_INTC	Interrupt flag clear register
DMA_CHxCTL (x=0..7)	Channel x control register
DMA_CHxCNT (x=0..7)	Channel x counter register
DMA_CHxPADDR (x=0..7)	Channel x peripheral base address register
DMA_CHxM0ADDR (x=0..7)	Channel x memory 0 base address register
DMA_CHxM1ADDR (x=0..7)	Channel x memory 1 base address register
DMA_CHxFCTL	Channel x FIFO control register

Registers	Descriptions
(x=0..7)	

### 3.9.2. Descriptions of Peripheral functions

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

**Table 3-179. DMA firmware function**

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

Function name	Function description
<code>dma_using_memory_get</code>	DMA using memory get
<code>dma_channel_subperipheral_select</code>	DMA channel peripheral select
<code>dma_flow_controller_config</code>	DMA flow controller configure
<code>dma_switch_buffer_mode_enable</code>	DMA flow controller enable
<code>dma_fifo_status_get</code>	DMA FIFO status get
<code>dma_flag_get</code>	check DMA flag is set or not
<code>dma_flag_clear</code>	clear DMA a channel flag
<code>dma_interrupt_flag_get</code>	check DMA flag is set or not
<code>dma_interrupt_flag_clear</code>	clear DMA a channel flag
<code>dma_interrupt_enable</code>	enable DMA interrupt
<code>dma_interrupt_disable</code>	disable DMA interrupt

### Structure `dma_multi_data_parameter_struct`

**Table 3-180. Structure `dma_multi_data_parameter_struct`**

Member name	Function description
<code>periph_addr</code>	peripheral base address
<code>periph_width</code>	transfer data size of peripheral
<code>periph_inc</code>	peripheral increasing mode
<code>memory0_addr</code>	memory 0 base address
<code>memory_width</code>	transfer data size of memory
<code>memory_inc</code>	memory increasing mode
<code>memory_burst_width</code>	memory burst width
<code>periph_burst_width</code>	periph burst width
<code>critical_value</code>	DMA circular value
<code>circular_mode</code>	DMA circular mode
<code>direction</code>	channel data transfer direction
<code>number</code>	channel transfer number
<code>priority</code>	channel priority level



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

**Table 3-181. Structure dma\_single\_data\_parameter\_struct**

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

## dma\_deinit

The description of dma\_deinit is shown as below:

**Table 3-182. Function dma\_deinit**

Function name	dma_deinit
Function prototype	void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	deinitialize DMA a channel registers
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0,1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..7)	DMA channel selection
Output parameter{out}	
-	-

Return value	
-	-

Example:

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

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

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

**Table 3-183. Function dma\_single\_data\_mode\_init**

Function name	dma_single_data_mode_init
Function prototype	void dma_single_data_mode_init(uint32_t dma_periph,dma_channel_enum channelx,dma_single_data_parameter_struct* init_struct);
Function descriptions	initialize DMA single data mode
Precondition	-
The called functions	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMAx(x=0,1)
Input parameter{in}	
<b>channelx</b>	specify which DMA channel is initialized
<i>DMA_CHx(x=0..7)</i>	DMA_CHx(x=0..7)
Input parameter{ in }	
<b>init_struct</b>	a dma_single_data_parameter_struct address, the structure members can refer to <a href="#">Table 3-181. Structure dma_single_data_parameter_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize DMA single data mode */
dma_single_data_parameter_struct dma_single_data_struct;
```

```
dma_single_data_mode_init(DMAx0, DMA_CH0, &dma_single_data_struct);
```

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

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

**Table 3-184. Function dma\_multi\_data\_mode\_init**

<b>Function name</b>	dma_multi_data_mode_init
<b>Function prototype</b>	void dma_multi_data_mode_init(uint32_t dma_periph,dma_channel_enum channelx,dma_multi_data_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize DMA multi data mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-184. Function dma_multi_data_mode_init</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize DMA multi data mode */
dma_multi_data_parameter_struct dma_multi_data_struct;
dma_init(DMA0, DMA_CH0, &dma_multi_data_struct);
```

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

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

Table 3-185. Function dma\_periph\_address\_config

Function name	dma_periph_address_config
Function prototype	void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
Function descriptions	set DMA peripheral base address
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0,1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..7)	DMA channel selection
Input parameter{in}	
address	peripheral base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### dma\_memory\_address\_config

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

Table 3-186. 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,uint8_t memory_flag,uint32_t address);
Function descriptions	set DMA memory base address

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>memory_flag</b>	DMA memory
<i>DMA_MEMORY_x(x=0,1)</i>	DMA memory selection
<b>Input parameter{in}</b>	
<b>address</b>	memory base address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
uint8_t g_destbuf[TRANSFER_NUM];
```

```
dma_memory_address_config(DMA0, DMA_CH0, DMA_MEMORY_0, (uint32_t) g_destbuf);
```

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

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

**Table 3-187. Function dma\_transfer\_number\_config**

<b>Function name</b>	dma_transfer_number_config
<b>Function prototype</b>	void dma_transfer_number_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t number);
<b>Function descriptions</b>	set the number of remaining data to be transferred by the DMA
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>number</b>	the number of remaining data to be transferred by the DMA
<i>0-0xffff</i>	number
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
#define TRANSFER_NUM                0x400

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

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

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

**Table 3-188. Function dma\_transfer\_number\_get**

<b>Function name</b>	dma_transfer_number_get
<b>Function prototype</b>	uint32_t dma_transfer_number_get(uint32_t dma_periph,dma_channel_enum channelx);
<b>Function descriptions</b>	get the number of remaining data to be transferred by the DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection

Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
<b>uint32_t(0-0xffff)</b>	the number of remaining data to be transferred by the DMA(0-0xffff)

Example:

```
uint32_t number = 0;
```

```
number = dma_transfer_number_get(DMA0, DMA_CH0);
```

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

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

**Table 3-189. 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>	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
<b>priority</b>	priority Level of this channel
<i>DMA_PRIORITY_LOW</i>	low priority
<i>DMA_PRIORITY_MEDI</i>	medium priority

<i>UM</i>	
<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\_burst\_beats\_config**

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

**Table 3-190. Function dma\_memory\_burst\_beats\_config**

<b>Function name</b>	dma_memory_burst_beats_config
<b>Function prototype</b>	void dma_memory_burst_beats_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t mbeat);
<b>Function descriptions</b>	configure transfer burst beats of memory
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>mbeat</b>	transfer burst beats
<i>DMA_MEMORY_BURST_SINGLE</i>	memory transfer single burst
<i>DMA_MEMORY_BURST_4BEAT</i>	memory transfer 4-beat burst



<i>T_4_BEAT</i>	
<i>DMA_MEMORY_BURS</i> <i>T_8_BEAT</i>	memory transfer 8-beat burst
<i>DMA_MEMORY_BURS</i> <i>T_16_BEAT</i>	memory transfer 16-beat burst
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
dma_memory_burst_beats_config (DMA0, DMA_CH0, DMA_MEMORY_BURST_8_BEAT);
```

### **dma\_periph\_burst\_beats\_config**

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

**Table 3-191. Function dma\_periph\_burst\_beats\_config**

<b>Function name</b>	dma_periph_burst_beats_config
<b>Function prototype</b>	void dma_periph_burst_beats_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t pbeat);
<b>Function descriptions</b>	configure transfer burst beats of peripheral
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>pbeat</b>	transfer burst beats
<i>DMA_PERIPH_BURST_SINGLE</i>	peripheral transfer single burst

<i>DMA_PERIPH_BURST_4_BEAT</i>	peripheral transfer 4-beat burst
<i>DMA_PERIPH_BURST_8_BEAT</i>	peripheral transfer 8-beat burst
<i>DMA_PERIPH_BURST_16_BEAT</i>	peripheral transfer 16-beat burst
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

**Table 3-192. Function dma\_memory\_width\_config**

<b>Function name</b>	dma_memory_width_config
<b>Function prototype</b>	void dma_memory_width_config(uint32_t dma_periph,dma_channel_enum channelx,uint32_t msize);
<b>Function descriptions</b>	configure transfer data size of memory
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>msize</b>	transfer data size of memory
<i>DMA_MEMORY_WIDT</i>	transfer data size of memory is 8-bit

<i>H_8BIT</i>	
<i>DMA_MEMORY_WIDT</i> <i>H_16BIT</i>	transfer data size of memory is 16-bit
<i>DMA_MEMORY_WIDT</i> <i>H_32BIT</i>	transfer data size 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 3-193. 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(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>psize</b>	transfer data size of peripheral
<i>DMA_PERIPHERAL_W</i> <i>IDTH_8BIT</i>	transfer data size of peripheral is 8-bit

<i>DMA_PERIPHERAL_WIDTH_16BIT</i>	transfer data size of peripheral is 16-bit
<i>DMA_PERIPHERAL_WIDTH_32BIT</i>	transfer data size 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\_address\_generation\_config**

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

**Table 3-194. Function dma\_memory\_address\_generation\_config**

<b>Function name</b>	dma_memory_address_generation_config
<b>Function prototype</b>	void dma_memory_address_generation_config(uint32_t dma_periph, dma_channel_enum channelx, uint8_t generation_algorithm);
<b>Function descriptions</b>	configure memory address generation generation_algorithm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>generation_algorithm</b>	the address generation algorithm
<i>DMA_MEMORY_INCR_EASE_ENABLE</i>	next address of memory is increasing address mode
<i>DMA_MEMORY_INCR</i>	next address of memory is fixed address mode

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

Example:

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

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

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

**Table 3-195. Function dma\_peripheral\_address\_generation\_config**

<b>Function name</b>	dma_peripheral_address_generation_config
<b>Function prototype</b>	void dma_peripheral_address_generation_config(uint32_t dma_periph,dma_channel_enum channelx,uint8_t generation_algorithm);
<b>Function descriptions</b>	configure peripheral address generation_algorithm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>generation_algorithm</b>	the address generation algorithm
<i>DMA_PERIPH_INCREASE_ENABLE</i>	next address of peripheral is increasing address mode
<i>DMA_PERIPH_INCREASE_DISABLE</i>	next address of peripheral is fixed address mode
<i>DMA_PERIPH_INCRE</i>	increasing steps of peripheral address is fixed

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

Example:

```
dma_peripheral_address_generation_config (DMA0, DMA_CH0, DMA_PERIPH_INCREASE_ENABLE);
```

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

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

**Table 3-196. 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>	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
DMAx(x=0,1)	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
DMA_CHx(x=0..7)	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_circulation_enable (DMA0, DMA_CH0);
```

## dma\_circulation\_disable

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

**Table 3-197. Function dma\_circulation\_disable**

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

Example:

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

## dma\_channel\_enable

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

**Table 3-198. Function dma\_channel\_enable**

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

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
dma_channel_enable (DMA0, DMA_CH0);
```

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

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

**Table 3-199. Function dma\_channel\_disable**

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



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

Example:

```
dma_channel_disable (DMA0, DMA_CH0);
```

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

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

**Table 3-200. 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,uint8_t direction);
<b>Function descriptions</b>	configure the direction of data transfer on the channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>direction</b>	specify the direction of data transfer
<i>DMA_PERIPH_TO_MEMORY</i>	read from peripheral and write to memory
<i>DMA_MEMORY_TO_PERIPH</i>	read from memory and write to peripheral
<i>DMA_MEMORY_TO_MEMORY</i>	read from memory and write to memory
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

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

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

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

**Table 3-201. Function dma\_switch\_buffer\_mode\_config**

Function name	dma_switch_buffer_mode_config
Function prototype	void dma_switch_buffer_mode_config(uint32_t dma_periph,dma_channel_enum channelx,uint32_t memory1_addr,uint32_t memory_select);
Function descriptions	DMA switch buffer mode config
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0,1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..7)	DMA channel selection
Input parameter{in}	
memory1_addr	memory1 base address
Input parameter{in}	
memory_select	DMA_MEMORY_0 or DMA_MEMORY_1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
dma_switch_buffer_mode_config(DMA0, DMA_CH0,0xaabbccdd,DMA_MEMORY_0);
```

## dma\_using\_memory\_get

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

**Table 3-202. Function dma\_using\_memory\_get**

<b>Function name</b>	dma_using_memory_get
<b>Function prototype</b>	uint32_t dma_using_memory_get(uint32_t dma_periph,dma_channel_enum channelx);
<b>Function descriptions</b>	DMA using memory get
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	the using memory

Example:

```
uint32_t using_memory = dma_using_memory_get(DMA0, DMA_CH0);
```

## dma\_channel\_subperipheral\_select

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

**Table 3-203. Function dma\_channel\_subperipheral\_select**

<b>Function name</b>	dma_channel_subperipheral_select
<b>Function prototype</b>	void dma_channel_subperipheral_select(uint32_t dma_periph,dma_channel_enum channelx,dma_subperipheral_enum sub_periph);

<b>Function descriptions</b>	DMA channel peripheral select
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>sub_periph</b>	specify DMA channel peripheral
<i>DMA_SUBPERIx(x=0..7)</i>	specify DMA channel peripheral select
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
dma_channel_subperipheral_select (DMA0, DMA_CH0, DMA_SUBPERI0);
```

### dma\_flow\_controller\_config

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

**Table 3-204. Function dma\_flow\_controller\_config**

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

<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>controller</b>	specify DMA flow controller
<i>DMA_FLOW_CONTROLLER_DMA</i>	DMA is the flow controller
<i>DMA_FLOW_CONTROLLER_PERI</i>	peripheral is the flow controller
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

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

**Table 3-205. Function dma\_switch\_buffer\_mode\_enable**

<b>Function name</b>	dma_switch_buffer_mode_enable
<b>Function prototype</b>	void dma_switch_buffer_mode_enable(uint32_t dma_periph,dma_channel_enum channelx,ControlStatus newvalue);
<b>Function descriptions</b>	DMA switch buffer mode enable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection

Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
<b>newvalue</b>	ENABLE or DISABLE
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

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

**Table 3-206. Function dma\_fifo\_status\_get**

<b>Function name</b>	dma_fifo_status_get
<b>Function prototype</b>	uint32_t dma_fifo_status_get(uint32_t dma_periph,dma_channel_enum channelx);
<b>Function descriptions</b>	DMA FIFO status get
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	

<b>uint32_t</b>	the using memory
-----------------	------------------

Example:

```
uint32_t using_memory dma_fifo_status_get (DMA0, DMA_CH0);
```

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

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

**Table 3-207. 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>	get DMA flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_FLAG_FEE</i>	FIFO error and exception flag
<i>DMA_FLAG_SDE</i>	single data mode exception flag
<i>DMA_FLAG_TAE</i>	transfer access error flag
<i>DMA_FLAG_HTF</i>	half transfer finish flag
<i>DMA_FLAG_FTF</i>	full transger finish flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
FlagStatus flag_status = dma_flag_get (DMA0, DMA_CH0, DMA_FLAG_FEE);
```

## **dma\_flag\_clear**

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

**Table 3-208. 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 DMA a channel flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
<i>DMA_FLAG_FEE</i>	FIFO error and exception flag
<i>DMA_FLAG_SDE</i>	single data mode exception flag
<i>DMA_FLAG_TAE</i>	transfer access error flag
<i>DMA_FLAG_HTF</i>	half transfer finish flag
<i>DMA_FLAG_FTF</i>	full transger finish flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:



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

### dma\_interrupt\_flag\_get

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

**Table 3-209. 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 interrupt);
<b>Function descriptions</b>	check DMA flag and interrupt enable bit is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which flag
<i>DMA_INT_FLAG_FEE</i>	FIFO error and exception flag
<i>DMA_INT_FLAG_SDE</i>	single data mode exception flag
<i>DMA_INT_FLAG_TAE</i>	transfer access error flag
<i>DMA_INT_FLAG_HTF</i>	half transfer finish flag
<i>DMA_INT_FLAG_FTF</i>	full transger finish flag
<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 3-210. 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 interrupt);
<b>Function descriptions</b>	clear DMA a channel interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which flag
<i>DMA_INT_FLAG_FEE</i>	FIFO error and exception flag
<i>DMA_INT_FLAG_SDE</i>	single data mode exception flag
<i>DMA_INT_FLAG_TAE</i>	transfer access error flag
<i>DMA_INT_FLAG_HTF</i>	half transfer finish flag
<i>DMA_INT_FLAG_FTF</i>	full transger finish flag
<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_FTF);
}
```

```
}

```

## **dma\_interrupt\_enable**

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

**Table 3-211. Function dma\_interrupt\_enable**

<b>Function name</b>	dma_interrupt_enable
<b>Function prototype</b>	void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
<b>Function descriptions</b>	enable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>source</b>	DMA interrupt source
<i>DMA_CHXCTL_SDEIE</i>	single data mode exception interrupt enable
<i>DMA_CHXCTL_TAEIE</i>	transfer access error interrupt enable
<i>DMA_CHXCTL_HTFIE</i>	half transfer finish interrupt enable
<i>DMA_CHXCTL_FTFIE</i>	full transfer finish interrupt enable
<i>DMA_CHXFCTL_FEEIE</i>	FIFO exception interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* DMA0 channel0 interrupt configuration */

```

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

## **dma\_interrupt\_disable**

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

**Table 3-212. Function dma\_interrupt\_disable**

<b>Function name</b>	dma_interrupt_disable
<b>Function prototype</b>	void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
<b>Function descriptions</b>	disable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dma_periph</b>	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>source</b>	DMA interrupt source
<i>DMA_CHXCTL_SDEIE</i>	single data mode exception interrupt enable
<i>DMA_CHXCTL_TAEIE</i>	transfer access error interrupt enable
<i>DMA_CHXCTL_HTFIE</i>	half transfer finish interrupt enable
<i>DMA_CHXCTL_FTFIE</i>	full transfer finish interrupt enable
<i>DMA_CHXFCTL_FEEIE</i>	FIFO exception interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* DMA0 channel0 interrupt configuration */
```

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

## 3.10. ENET

There is a media access controller (MAC) designed in Ethernet module to support 10/100Mbps interface speed. For more efficient data transfer between Ethernet and memory, a DMA controller is designed in this module. The support interface protocol for Ethernet is media independent interface (MII) and reduced media independent interface (RMII). The ENET registers are listed in chapter [3.10.1](#), the ENET firmware functions are introduced in chapter [3.10.2](#).

### 3.10.1. Descriptions of Peripheral registers

ENET registers are listed in the table shown as below:

**Table 3-213. ENET Registers**

Registers	Descriptions
ENET_MAC_CFG	MAC configuration register
ENET_MAC_FRMF	MAC frame filter register
ENET_MAC_HLH	MAC hash list high register
ENET_MAC_HLL	MAC hash list low register
ENET_MAC_PHY_CTL	MAC PHY control register
ENET_MAC_PHY_DATA	MAC PHY data register
ENET_MAC_FCTL	MAC flow control register
ENET_MAC_VLT	MAC VLAN tag register
ENET_MAC_RWFF	MAC remote wakeup frame filter register
ENET_MAC_WUM	MAC wakeup management register
ENET_MAC_DBG	MAC debug register
ENET_MAC_INTF	MAC interrupt flag register
ENET_MAC_INTMSK	MAC interrupt mask register
ENET_MAC_ADDR0H	MAC address 0 high register
ENET_MAC_ADDR0L	MAC address 0 low register

Registers	Descriptions
0L	
ENET_MAC_ADDR 1H	MAC address 1 high register
ENET_MAC_ADDR 1L	MAC address 1 low register
ENET_MAC_ADDT 2H	MAC address 2 high register
ENET_MAC_ADDR 2L	MAC address 2 low register
ENET_MAC_ADDR 3H	MAC address 3 high register
ENET_MAC_ADDR 3L	MAC address 3 low register
ENET_MAC_FCTH	MAC flow control threshold register
ENET_MSC_CTL	MSC control register
ENET_MSC_RINTF	MSC receive interrupt flag register
ENET_MSC_TINTF	MSC transmit interrupt flag register
ENET_MSC_RINT MSK	MSC receive interrupt mask register
ENET_MSC_TINTM SK	MSC transmit interrupt mask register
ENET_MSC_SCCN T	MSC transmitted good frames after a single collision counter register
ENET_MSC_MSCC NT	MSC transmitted good frames after more than a single collision counter register
ENET_MSC_TGFC NT	MSC transmitted good frames counter register
ENET_MSC_RFCE CNT	MSC received frames with CRC error counter register
ENET_MSC_RFAE CNT	MSC received frames with alignment error counter register
ENET_MSC_RGUF CNT	MSC received good unicast frames counter register

Registers	Descriptions
ENET_PTP_TSCTL	PTP time stamp control register
ENET_PTP_SSINC	PTP subsecond increment register
ENET_PTP_TSH	PTP time stamp high register
ENET_PTP_TSL	PTP time stamp low register
ENET_PTP_TSUH	PTP time stamp update high register
ENET_PTP_TSUL	PTP time stamp update low register
ENET_PTP_TSADD END	PTP time stamp addend register
ENET_PTP_ETH	PTP expected time high register
ENET_PTP_ETL	PTP expected time low register
ENET_PTP_TSF	PTP time stamp flag register
ENET_PTP_PPSCTL	PTP PPS control register
ENET_DMA_BCTL	DMA bus control register
ENET_DMA_TPEN	DMA transmit poll enable register
ENET_DMA_RPEN	DMA receive poll enable register
ENET_DMA_RDTA DDR	DMA receive descriptor table address register
ENET_DMA_TDTA DDR	DMA transmit descriptor table address register
ENET_DMA_STAT	DMA status register
ENET_DMA_CTL	DMA control register
ENET_DMA_INTEN	DMA interrupt enable register
ENET_DMA_MFBO CNT	DMA missed frame and buffer overflow counter register
ENET_DMA_RSWD C	DMA receive state watchdog counter register
ENET_DMA_CTDA DDR	DMA current transmit descriptor address register
ENET_DMA_CRDA	DMA current receive descriptor address register

Registers	Descriptions
DDR	
ENET_DMA_CTBA DDR	DMA current transmit buffer address register
ENET_DMA_CRBA DDR	DMA current receive buffer address register

### 3.10.2. Descriptions of Peripheral functions

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

**Table 3-214. ENET firmware function**

Function name	Function description
main function	
enet_deinit	deinitialize the ENET, and reset structure parameters for ENET initialization
enet_initpara_config	configure the parameters which are usually less cared for initialization
enet_init	initialize ENET peripheral with generally concerned parameters and the less cared parameters
enet_software_reset	reset all core internal registers located in CLK_TX and CLK_RX
enet_rxframe_size_get	check receive frame valid and return frame size
enet_descriptors_chain_init	initialize the dma tx/rx descriptors's parameters in chain mode
enet_descriptors_ring_init	initialize the dma tx/rx descriptors's parameters in ring mode
enet_frame_receive	handle current received frame data to application buffer
enet_frame_transmit	handle application buffer data to transmit it
enet_transmit_checksum_config	configure the transmit IP frame checksum offload calculation and insertion
enet_enable	ENET Tx and Rx function enable (include MAC and DMA module)
enet_disable	ENET Tx and Rx function disable (include MAC and DMA module)
enet_mac_address_set	configure MAC address
enet_mac_address_get	get MAC address



Function name	Function description
enet_flag_get	get the ENET MAC/MSD/PTP/DMA status flag
enet_flag_clear	clear the ENET DMA status flag
enet_interrupt_enable	enable ENET MAC/MSD/DMA interrupt
enet_interrupt_disable	disable ENET MAC/MSD/DMA interrupt
enet_interrupt_flag_get	get ENET MAC/MSD/DMA interrupt flag
enet_interrupt_flag_clear	clear ENET DMA interrupt flag
MAC function	
enet_tx_enable	ENET Tx function enable (include MAC and DMA module)
enet_tx_disable	ENET Tx function disable (include MAC and DMA module)
enet_rx_enable	ENET Rx function enable (include MAC and DMA module)
enet_rx_disable	ENET Rx function disable (include MAC and DMA module)
enet_registers_get	put registers value into the application buffer
enet_debug_status_get	get the enet debug status from the debug register
enet_address_filter_enable	enable the MAC address filter
enet_address_filter_disable	disable the MAC address filter
enet_address_filter_config	configure the MAC address filter
enet_phy_config	PHY interface configuration (configure SMI clock and reset PHY chip)
enet_phy_write_read	write to/read from a PHY register
enet_phyloopback_enable	enable the loopback function of phy chip
enet_phyloopback_disable	disable the loopback function of phy chip
enet_forward_feature_enable	enable ENET forward feature
enet_forward_feature_disable	disable ENET forward feature
enet_fliter_feature_enable	enable ENET fliter feature
enet_fliter_feature_disable	disable ENET fliter feature
flow control function	
enet_pauseframe_generate	generate the pause frame, ENET will send pause frame after enable transmit flow control

Function name	Function description
enet_pauseframe_detect_config	configure the pause frame detect type
enet_pauseframe_config	configure the pause frame parameters
enet_flowcontrol_threshold_config	configure the threshold of the flow control(deactive and active threshold)
enet_flowcontrol_feature_enable	enable ENET flow control feature
enet_flowcontrol_feature_disable	disable ENET flow control feature
DMA function	
enet_dmaprocess_state_get	get the dma transmit/receive process state
enet_dmaprocess_resume	poll the dma transmission/reception enable
enet_rxprocess_check_recovery	check and recover the Rx process
enet_txfifo_flush	flush the ENET transmit fifo, and wait until the flush operation completes
enet_current_desc_address_get	get the transmit/receive address of current descriptor, or current buffer, or descriptor table
enet_desc_information_get	get the Tx or Rx descriptor information
enet_missed_frame_counter_get	get the number of missed frames during receiving
descriptor function	
enet_desc_flag_get	get the bit flag of ENET dma descriptor
enet_desc_flag_set	set the bit flag of ENET dma tx descriptor
enet_desc_flag_clear	clear the bit flag of ENET dma tx descriptor
enet_rx_desc_immediate_receive_complete_interrupt	when receiving the completed, set RS bit in ENET_DMA_STAT register will immediately set
enet_rx_desc_delay_receive_complete_interrupt	when receiving the completed, set RS bit in ENET_DMA_STAT register will is set after a configurable delay time
enet_rxframe_drop	drop current receive frame
enet_dma_feature_enable	enable DMA feature
enet_dma_feature_disable	disable DMA feature
enhanced descriptor	
enet_rx_desc_enhanced_status_get	get the bit of extended status flag in ENET DMA descriptor

Function name	Function description
enet_desc_select_enhanced_mode	configure descriptor to work in enhanced mode
enet_ptp_enhanced_descriptors_chain_init	initialize the dma Tx/Rx descriptors's parameters in enhanced chain mode with ptp function
enet_ptp_enhanced_descriptors_ring_init	initialize the dma Tx/Rx descriptors's parameters in enhanced ring mode with ptp function
enet_ptpframe_receive_enhanced_mode	receive a packet data with timestamp values to application buffer, when the DMA is in enhanced mode
enet_ptpframe_transmit_enhanced_mode	send data with timestamp values in application buffer as a transmit packet, when the DMA is in enhanced mode
normal descriptor	
enet_desc_select_normal_mode	configure descriptor to work in normal mode
enet_ptp_normal_descriptors_chain_init	initialize the dma Tx/Rx descriptors's parameters in normal chain mode with ptp function
enet_ptp_normal_descriptors_ring_init	initialize the dma Tx/Rx descriptors's parameters in normal ring mode with ptp function
enet_ptpframe_receive_normal_mode	receive a packet data with timestamp values to application buffer, when the DMA is in normal mode
enet_ptpframe_transmit_normal_mode	send data with timestamp values in application buffer as a transmit packet, when the DMA is in normal mode
WUM function	
enet_wum_filter_register_pointer_reset	wakeup frame filter register pointer reset
enet_wum_filter_config	set the remote wakeup frame registers
enet_wum_feature_enable	enable wakeup management features
enet_wum_feature_disable	disable wakeup management features
MSC function	
enet_msc_counters_reset	reset the MAC statistics counters
enet_msc_feature_enable	enable the MAC statistics counter features
enet_msc_feature_disable	disable the MAC statistics counter features
enet_msc_counters_preset_config	configure MAC statistics counters preset mode
enet_msc_counters_get	get MAC statistics counter

Function name	Function description
PTP function	
enet_ptp_feature_enable	enable the PTP features
enet_ptp_feature_disable	disable the PTP features
enet_ptp_timestamp_function_config	configure the PTP timestamp function
enet_ptp_subsecond_increment_config	configure the PTP system time subsecond increment value
enet_ptp_timestamp_addend_config	adjusting the PTP clock frequency only in fine update mode
enet_ptp_timestamp_update_config	initializing or adding/subtracting to second of the PTP system time
enet_ptp_expected_time_config	configure the PTP expected target time
enet_ptp_system_time_get	get the PTP current system time
enet_ptp_pps_output_frequency_config	configure the PPS output frequency
Other	
enet_initpara_reset	reset the ENET initpara struct, call it before using enet_initpara_config()

### Structure enet\_initpara\_struct

**Table 3-215. Structure enet\_initpara\_struct**

member name	Function description
option_enable	select which function to configure
forward_frame	frame forward related parameters
dmabus_mode	DMA bus mode related parameters
dma_maxburst	DMA max burst related parameters
dma_arbitration	DMA Tx and Rx arbitration related parameters
store_forward_mode	store forward mode related parameters
dma_function	DMA control related parameters
vlan_config	VLAN tag related parameters
flow_control	flow control related parameters
hashtable_high	hash list high 32-bit related parameters

hashtable_low	hash list low 32-bit related parameters
framesfilter_mode	frame filter control related parameters
halfduplex_param	halfduplex related parameters
timer_config	frame timer related parameters
interframegap	inter frame gap related parameters

### Structure `enet_descriptors_struct`

**Table 3-216. Structure `enet_descriptors_struct`**

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

### Structure `enet_ptp_systime_struct`

**Table 3-217. Structure `enet_ptp_systime_struct`**

member name	Function description
second	second of system time
nanosecond	nanosecond of system time
sign	sign of system time

### `enet_deinit`

The description of `enet_deinit` is shown as below:

**Table 3-218. Function `enet_deinit`**

Function name	<code>enet_deinit</code>
Function prototype	<code>void enet_deinit(void);</code>

<b>Function descriptions</b>	deinitialize the ENET, and reset structure parameters for ENET initialization
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable()/rcu_periph_reset_disable()/enet_initpara_reset()
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* deinitialize the ENET */
enet_deinit( );
```

### enet\_initpara\_config

The description of enet\_initpara\_config is shown as below:

**Table 3-219. Function enet\_initpara\_config**

<b>Function name</b>	enet_initpara_config
<b>Function prototype</b>	void enet_initpara_config(enet_option_enum option, uint32_t para)
<b>Function descriptions</b>	configure the parameters which are usually less cared for initialization, this function must be called before enet_init(), otherwise configuration will be no effect
<b>Precondition</b>	enet_initpara_reset(void)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>option</b>	different function option, which is related to several parameters only one parameter can be selected which is shown as below
<i>FORWARD_OPTION</i>	choose to configure the frame forward related parameters
<i>DMABUS_OPTION</i>	choose to configure the DMA bus mode related parameters
<i>DMA_MAXBURST_OPTION</i>	choose to configure the DMA max burst related parameters
<i>DMA_ARBITRATION_</i>	choose to configure the DMA arbitration related parameters

OPTION	
STORE_OPTION	choose to configure the store forward mode related parameters
DMA_OPTION	choose to configure the DMA related parameters
VLAN_OPTION	choose to configure vlan related parameters
FLOWCTL_OPTION	choose to configure flow control related parameters
HASHH_OPTION	choose to configure hash high
HASHL_OPTION	choose to configure hash low
FILTER_OPTION	choose to configure frame filter related parameters
HALFDUPLEX_OPTION	choose to configure halfduplex mode related parameters
TIMER_OPTION	choose to configure time counter related parameters
INTERFRAMEGAP_OPTION	choose to configure the inter frame gap related parameters
Input parameter{in}	
para (the value according to the parameter <b>option</b> )	all the related values should be configured which are shown as below example: <b>para</b> = (value1   value2   value3...)
When value of parameter <b>option</b> is FORWARD_OPTION	
value1	ENET_AUTO_PADCRC_DROP_ENABLE / ENET_AUTO_PADCRC_DROP_DISABLE
value2	ENET_FORWARD_ERRFRAMES_ENABLE / ENET_FORWARD_ERRFRAMES_DISABLE
value3	ENET_FORWARD_UNDEFSZ_GOODFRAMES_ENABLE / ENET_FORWARD_UNDEFSZ_GOODFRAMES_DISABLE
value4	ENET_FORWARD_UNDEFSZ_GOODFRAMES_ENABLE / ENET_FORWARD_UNDEFSZ_GOODFRAMES_DISABLE
When value of parameter <b>option</b> is DMABUS_OPTION	
value1	ENET_ADDRESS_ALIGN_ENABLE / ENET_ADDRESS_ALIGN_DISABLE
value2	ENET_FIXED_BURST_ENABLE / ENET_FIXED_BURST_DISABLE
value3	ENET_MIXED_BURST_ENABLE/ ENET_MIXED_BURST_DISABLE
When value of parameter <b>option</b> is DMA_MAXBURST_OPTION	
value1	ENET_RXDP_1BEAT / ENET_RXDP_2BEAT/ ENET_RXDP_4BEAT /

	<i>ENET_RXDP_8BEAT / ENET_RXDP_16BEAT / ENET_RXDP_32BEAT / ENET_RXDP_4xPGBL_4BEAT / ENET_RXDP_4xPGBL_8BEAT / ENET_RXDP_4xPGBL_16BEAT / ENET_RXDP_4xPGBL_32BEAT / ENET_RXDP_4xPGBL_64BEAT / ENET_RXDP_4xPGBL_128BEAT</i>
value2	<i>ENET_PGBL_1BEAT / ENET_PGBL_2BEAT / ENET_PGBL_4BEAT / ENET_PGBL_8BEAT / ENET_PGBL_16BEAT / ENET_PGBL_32BEAT / ENET_PGBL_4xPGBL_4BEAT / ENET_PGBL_4xPGBL_8BEAT / ENET_PGBL_4xPGBL_16BEAT / ENET_PGBL_4xPGBL_32BEAT / ENET_PGBL_4xPGBL_64BEAT / ENET_PGBL_4xPGBL_128BEAT</i>
value3	<i>ENET_RXTX_DIFFERENT_PGBL / ENET_RXTX_SAME_PGBL</i>
When value of parameter <b>option</b> is <i>DMA_ARBITRATION_OPTION</i>	
value1	<i>ENET_ARBITRATION_RXPRIORTX</i>
value2	<i>ENET_ARBITRATION_RXTX_1_1 / ENET_ARBITRATION_RXTX_2_1 / ENET_ARBITRATION_RXTX_3_1 / ENET_ARBITRATION_RXTX_4_1</i>
When value of parameter <b>option</b> is <i>STORE_OPTION</i>	
value1	<i>ENET_RX_MODE_STOREFORWARD / ENET_RX_MODE_CUTTHROUGH</i>
value2	<i>ENET_TX_MODE_STOREFORWARD / ENET_TX_MODE_CUTTHROUGH</i>
value3	<i>ENET_RX_THRESHOLD_64BYTES / ENET_RX_THRESHOLD_32BYTES / ENET_RX_THRESHOLD_96BYTES / ENET_RX_THRESHOLD_128BYTES</i>
value4	<i>ENET_TX_THRESHOLD_64BYTES / ENET_TX_THRESHOLD_128BYTES / ENET_TX_THRESHOLD_192BYTES / ENET_TX_THRESHOLD_256BYTES / ENET_TX_THRESHOLD_40BYTES / ENET_TX_THRESHOLD_32BYTES / ENET_TX_THRESHOLD_24BYTES / ENET_TX_THRESHOLD_16BYTES</i>
When value of parameter <b>option</b> is <i>DMA_OPTION</i>	
value1	<i>ENET_FLUSH_RXFRAME_ENABLE / ENET_FLUSH_RXFRAME_DISABLE</i>
value2	<i>ENET_SECONDFRAME_OPT_ENABLE / ENET_SECONDFRAME_OPT_DISABLE</i>
value3	<i>ENET_ENHANCED_DESCRIPTOR/ ENET_NORMAL_DESCRIPTOR</i>
When value of parameter <b>option</b> is <i>VLAN_OPTION</i>	
value1	<i>ENET_VLANTAGCOMPARISON_12BIT/ ENET_VLANTAGCOMPARISON_16BIT</i>
value2	<i>MAC_VLT_VLTI(regval)</i>



When value of parameter <b>option</b> is <i>FLOWCTL_OPTION</i>	
value1	<i>MAC_FCTL_PTM(regval)</i>
value2	<i>ENET_ZERO_QUANTA_PAUSE_ENABLE /</i> <i>ENET_ZERO_QUANTA_PAUSE_DISABLE</i>
value3	<i>ENET_PAUSETIME_MINUS4 / ENET_PAUSETIME_MINUS28 /</i> <i>ENET_PAUSETIME_MINUS144/ENET_PAUSETIME_MINUS256</i>
value4	<i>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDetect /</i> <i>ENET_UNIQUE_PAUSEDetect</i>
value5	<i>ENET_RX_FLOWCONTROL_ENABLE /</i> <i>ENET_RX_FLOWCONTROL_DISABLE</i>
value6	<i>ENET_TX_FLOWCONTROL_ENABLE /</i> <i>ENET_TX_FLOWCONTROL_DISABLE</i>
When value of parameter <b>option</b> is <i>HASHH_OPTION</i>	
value1	<i>0x0~0xFFFF FFFFU</i>
When value of parameter <b>option</b> is <i>HASHL_OPTION</i>	
value1	<i>0x0~0xFFFF FFFFU</i>
When value of parameter <b>option</b> is <i>FILTER_OPTION</i>	
value1	<i>ENET_SRC_FILTER_NORMAL_ENABLE /</i> <i>ENET_SRC_FILTER_INVERSE_ENABLE / ENET_SRC_FILTER_DISABLE</i>
value2	<i>ENET_DEST_FILTER_INVERSE_ENABLE /</i> <i>ENET_DEST_FILTER_INVERSE_DISABLE</i>
value3	<i>ENET_MULTICAST_FILTER_HASH_OR_PERFECT /</i> <i>ENET_MULTICAST_FILTER_HASH /</i> <i>ENET_MULTICAST_FILTER_PERFECT /</i> <i>ENET_MULTICAST_FILTER_NONE</i>
value4	<i>ENET_UNICAST_FILTER_EITHER / ENET_UNICAST_FILTER_HASH /</i> <i>ENET_UNICAST_FILTER_PERFECT</i>
value5	<i>ENET_PCFRM_PREVENT_ALL /</i> <i>ENET_PCFRM_PREVENT_PAUSEFRAME /</i> <i>ENET_PCFRM_FORWARD_ALL / ENET_PCFRM_FORWARD_FILTERED</i>
When value of parameter <b>option</b> is <i>HALFDUPLEX_OPTION</i>	
value1	<i>ENET_CARRIERSENSE_ENABLE / ENET_CARRIERSENSE_DISABLE</i>
value2	<i>ENET_RECEIVEOWN_ENABLE / ENET_RECEIVEOWN_DISABLE</i>

value3	<i>ENET_RETRYTRANSMISSION_ENABLE / ENET_RETRYTRANSMISSION_DISABLE</i>
value4	<i>ENET_BACKOFFLIMIT_10 / ENET_BACKOFFLIMIT_8 / ENET_BACKOFFLIMIT_4 / ENET_BACKOFFLIMIT_1</i>
value5	<i>ENET_DEFERRALCHECK_ENABLE / ENET_DEFERRALCHECK_DISABLE</i>
When value of parameter <b>option</b> is <i>TIMER_OPTION</i>	
value1	<i>ENET_WATCHDOG_ENABLE / ENET_WATCHDOG_DISABLE</i>
value2	<i>ENET_JABBER_ENABLE / ENET_JABBER_DISABLE</i>
When value of parameter <b>option</b> is <i>INTERFRAMEGAP_OPTION</i>	
value1	<i>ENET_INTERFRAMEGAP_96BIT / ENET_INTERFRAMEGAP_88BIT / ENET_INTERFRAMEGAP_80BIT / ENET_INTERFRAMEGAP_72BIT / ENET_INTERFRAMEGAP_64BIT / ENET_INTERFRAMEGAP_56BIT / ENET_INTERFRAMEGAP_48BIT / ENET_INTERFRAMEGAP_40BIT</i>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config DMA option of the ENET */
```

```
enet_initpara_reset();
```

```
enet_initpara_config(DMA_OPTION, ENET_FLUSH_RXFRAME_ENABLE | ENET_SECONDFRAME_OPT_ENABLE | ENET_NORMAL_DESCRIPTOR);
```

## enet\_init

The description of enet\_init is shown as below:

**Table 3-220. Function enet\_init**

<b>Function name</b>	enet_init
<b>Function prototype</b>	ErrStatus enet_init(enet_mediamode_enum mediamode, enet_chksumconf_enum checksum, enet_frmrecept_enum recept);
<b>Function descriptions</b>	initialize ENET peripheral with generally concerned parameters and the less cared parameters
<b>Precondition</b>	enet_deinit ()

The called functions	enet_phy_config /enet_phy_write_read
Input parameter{in}	
<b>mediamode</b>	PHY mode and mac loopback configurations, only one parameter can be selected
<i>ENET_AUTO_NEGOTIATION</i>	PHY auto negotiation
<i>ENET_100M_FULLDUPLEX</i>	100Mbit/s, full-duplex
<i>ENET_100M_HALFDUPLEX</i>	100Mbit/s, half-duplex
<i>ENET_10M_FULLDUPLEX</i>	10Mbit/s, full-duplex
<i>ENET_10M_HALFDUPLEX</i>	10Mbit/s, half-duplex
<i>ENET_LOOPBACKMODE</i>	MAC in loopback mode at the MII
Input parameter{in}	
<b>checksum</b>	IP frame checksum offload function, only one parameter can be selected
<i>ENET_NO_AUTOCHECKSUM</i>	disable IP frame checksum function
<i>ENET_AUTOCHECKSUM_DROP_FAILFRAMES</i>	enable IP frame checksum function
<i>ENET_AUTOCHECKSUM_ACCEPT_FAILFRAMES</i>	enable IP frame checksum function, and the received frame with only payload error but no other errors will not be dropped
Input parameter{in}	
<b>recept</b>	frame filter function, only one parameter can be selected
<i>ENET_PROMISCUOUS_MODE</i>	promiscuous mode enabled
<i>ENET_RECEIVEALL</i>	all received frame are forwarded to application
<i>ENET_BROADCAST_FILTERS_PASS</i>	the address filters pass all received broadcast frames
<i>ENET_BROADCAST_FILTERS_FILTER</i>	the address filters filter all incoming broadcast frames

<i>RAMES_DROP</i>	
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* initialize ENET peripheral */
```

```
ErrStatus enet_init_status;
```

```
enet_init_status = enet_init(ENET_AUTO_NEGOTIATION, ENET_AUTOCHECKSUM_DROP_FAILFRAMES, ENET_BROADCAST_FRAMES_PASS);
```

### enet\_software\_reset

The description of enet\_software\_reset is shown as below:

**Table 3-221. Function enet\_software\_reset**

<b>Function name</b>	enet_software_reset
<b>Function prototype</b>	ErrStatus enet_software_reset(void);
<b>Function descriptions</b>	reset all core internal registers located in CLK_TX and CLK_RX
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* reset all core internal registers located in CLK_TX and CLK_RX */
```

```
ErrStatus reval_state = ERROR;
```

```
reval_state = enet_software_reset();
```

## enet\_rxframe\_size\_get

The description of enet\_rxframe\_size\_get is shown as below:

**Table 3-222. Function enet\_rxframe\_size\_get**

<b>Function name</b>	enet_rxframe_size_get
<b>Function prototype</b>	uint32_t enet_rxframe_size_get(void);
<b>Function descriptions</b>	check receive frame valid and return frame size
<b>Precondition</b>	-
<b>The called functions</b>	enet_rxframe_drop()
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	size of received frame 0x0 - 0x3FFF

Example:

```
/* check receive frame valid */
uint32_t reval;
reval = enet_rxframe_size_get();
```

## enet\_descriptors\_chain\_init

The description of enet\_descriptors\_chain\_init is shown as below:

**Table 3-223. Function enet\_descriptors\_chain\_init**

<b>Function name</b>	enet_descriptors_chain_init
<b>Function prototype</b>	void enet_descriptors_chain_init(enet_dmadirection_enum direction);
<b>Function descriptions</b>	initialize the DMA Tx/Rx descriptors's parameters in chain mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>direction</b>	the descriptors which users want to init only one parameter can be selected which is shown as below

<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
enet_descriptors_chain_init(ENET_DMA_TX);
```

### enet\_descriptors\_ring\_init

The description of enet\_descriptors\_ring\_init is shown as below:

**Table 3-224. Function enet\_descriptors\_ring\_init**

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

Example:

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

```
enet_descriptors_ring_init(ENET_DMA_TX);
```

## enet\_frame\_receive

The description of enet\_frame\_receive is shown as below:

**Table 3-225. Function enet\_frame\_receive**

<b>Function name</b>	enet_frame_receive
<b>Function prototype</b>	ErrStatus enet_frame_receive(uint8_t *buffer, uint32_t bufsize);
<b>Function descriptions</b>	handle current received frame data to application buffer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>bufsize</b>	the size of buffer which is the parameter in function, (0 -- 1524)
<b>Output parameter{out}</b>	
<b>buffer</b>	pointer to the received frame data if the input is NULL, user should copy data in application by himself
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* transfer received frame data to application buffer */
```

```
uint8_t data_buffer[1500];
```

```
uint32_t data_size;
```

```
enet_frame_receive(data_buffer, &data_size);
```

## enet\_frame\_transmit

The description of enet\_frame\_transmit is shown as below:

**Table 3-226. Function enet\_frame\_transmit**

<b>Function name</b>	enet_frame_transmit
<b>Function prototype</b>	ErrStatus enet_frame_transmit(uint8_t *buffer, uint32_t length);
<b>Function descriptions</b>	handle application buffer data to transmit it
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>buffer</b>	pointer to the frame data to be transmitted if the input is NULL, user should handle the data in application by himself
<b>Input parameter{in}</b>	
<b>length</b>	the length of frame data to be transmitted, (0 -- 1524)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* transfer buffer data of application */
uint8_t data_buffer[1500];
uint32_t data_size = 800;
enet_frame_transmit (data_buffer, data_size);
```

### enet\_transmit\_checksum\_config

The description of enet\_transmit\_checksum\_config is shown as below:

**Table 3-227. Function enet\_transmit\_checksum\_config**

<b>Function name</b>	enet_transmit_checksum_config
<b>Function prototype</b>	void enet_transmit_checksum_config(enet_descriptors_struct *desc, uint32_t checksum);
<b>Function descriptions</b>	configure the transmit IP frame checksum offload calculation and insertion
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>desc</b>	the descriptor pointer which users want to configure, the structure members can refer to <a href="#">Structure enet_descriptors_struct</a>
<b>Input parameter{in}</b>	
<b>checksum</b>	IP frame checksum configuration only one parameter can be selected which is shown as below
<b>ENET_CHECKSUM_DISABLE</b>	checksum insertion disabled



<i>ENET_CHECKSUM_IP V4HEADER</i>	only IP header checksum calculation and insertion are enabled
<i>ENET_CHECKSUM_T CPUDPICMP_SEGME NT</i>	TCP/UDP/ICMP checksum insertion calculated but pseudo-header
<i>ENET_CHECKSUM_T CPUDPICMP_FULLL</i>	TCP/UDP/ICMP checksum insertion fully calculated
<b>Output parameter{out}</b>	
<b>Return value</b>	

Example:

```
/* configure the transmit IP frame checksum offload calculation and insertion */
enet_descriptors_struct rx_desc;
enet_transmit_checksum_config(rx_desc, ENET_CHECKSUM_TCPUDPICMP_FULLL);
```

## enet\_enable

The description of enet\_enable is shown as below:

**Table 3-228. Function enet\_enable**

<b>Function name</b>	enet_enable
<b>Function prototype</b>	void enet_enable(void);
<b>Function descriptions</b>	ENET Tx and Rx function enable (include MAC and DMA module)
<b>Precondition</b>	-
<b>The called functions</b>	enet_tx_enable() /enet_rx_enable()
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the ENET */
```

```
enet_enable();
```

### enet\_disable

The description of enet\_disable is shown as below:

**Table 3-229. Function enet\_disable**

<b>Function name</b>	enet_disable
<b>Function prototype</b>	void enet_disable(void);
<b>Function descriptions</b>	ENET Tx and Rx function disable (include MAC and DMA module)
<b>Precondition</b>	-
<b>The called functions</b>	enet_tx_disable() /enet_rx_disable()
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the ENET */
```

```
enet_disable();
```

### enet\_mac\_address\_set

The description of enet\_mac\_address\_set is shown as below:

**Table 3-230. Function enet\_mac\_address\_set**

<b>Function name</b>	enet_mac_address_set
<b>Function prototype</b>	void enet_mac_address_set(enet_macaddress_enum mac_addr, uint8_t paddr[]);
<b>Function descriptions</b>	configure MAC address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>mac_addr</b>	select which MAC address will be set only one parameter can be selected which is shown as below
<i>ENET_MAC_ADDRES</i> <i>S0</i>	set MAC address 0 filter
<i>ENET_MAC_ADDRES</i> <i>S1</i>	set MAC address 1 filter
<i>ENET_MAC_ADDRES</i> <i>S2</i>	set MAC address 2 filter
<i>ENET_MAC_ADDRES</i> <i>S3</i>	set MAC address 3 filter
<b>Input parameter{in}</b>	
<b>paddr</b>	the buffer pointer which stores the MAC address little-ending store, such as MAC address is aa:bb:cc:dd:ee:22, the buffer is {22, ee, dd, cc, bb, aa}
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* config mac address */
netif->hwaddr[0] = 0x02;
netif->hwaddr[1] = 0xaa;
netif->hwaddr[2] = 0xbb;
netif->hwaddr[3] = 0xcc;
netif->hwaddr[4] = 0xdd;
netif->hwaddr[5] = 0xee;

enet_mac_address_set(ENET_MAC_ADDRESS0, netif->hwaddr);

```

### enet\_mac\_address\_get

The description of enet\_mac\_address\_get is shown as below:

**Table 3-231. Function enet\_mac\_address\_get**

<b>Function name</b>	enet_mac_address_get
----------------------	----------------------

<b>Function prototype</b>	void enet_mac_address_get(enet_macaddress_enum mac_addr, uint8_t paddr[]);
<b>Function descriptions</b>	get MAC address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mac_addr</b>	select which MAC address will be set only one parameter can be selected which is shown as below
ENET_MAC_ADDRES S0	set MAC address 0 filter
ENET_MAC_ADDRES S1	set MAC address 1 filter
ENET_MAC_ADDRES S2	set MAC address 2 filter
ENET_MAC_ADDRES S3	set MAC address 3 filter
<b>Output parameter{out}</b>	
<b>paddr</b>	the buffer pointer which stores the MAC address little-ending store, such as MAC address is aa:bb:cc:dd:ee:22, the buffer is {22, ee, dd, cc, bb, aa}
<b>Return value</b>	
-	-

Example:

```
/* get mac address */
```

```
enet_mac_address_get (ENET_MAC_ADDRESS0, netif->hwaddr);
```

### enet\_flag\_get

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

**Table 3-232. Function enet\_flag\_get**

<b>Function name</b>	enet_flag_get
<b>Function prototype</b>	FlagStatus enet_flag_get(enet_flag_enum enet_flag);
<b>Function descriptions</b>	get the ENET MAC/MSC/PTP/DMA status flag

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>enet_flag</b>	ENET status flag, refer to enet_flag_enum only one parameter can be selected which is shown as below
<i>ENET_MAC_FLAG_MP KR</i>	magic packet received flag
<i>ENET_MAC_FLAG_W UFR</i>	wakeup frame received flag
<i>ENET_MAC_FLAG_FL OWCONTROL</i>	flow control status flag
<i>ENET_MAC_FLAG_W UM</i>	WUM status flag
<i>ENET_MAC_FLAG_MS C</i>	MSC status flag
<i>ENET_MAC_FLAG_MS CR</i>	MSC receive status flag
<i>ENET_MAC_FLAG_MS CT</i>	MSC transmit status flag
<i>ENET_MAC_FLAG_TM ST</i>	time stamp trigger status flag
<i>ENET_PTP_FLAG_TS SCO</i>	timestamp second counter overflow flag
<i>ENET_PTP_FLAG_TT M</i>	target time match flag
<i>ENET_MSC_FLAG_RF CE</i>	received frames CRC error flag
<i>ENET_MSC_FLAG_RF AE</i>	received frames alignment error flag
<i>ENET_MSC_FLAG_RG UF</i>	received good unicast frames flag
<i>ENET_MSC_FLAG_TG FSC</i>	transmitted good frames single collision flag
<i>ENET_MSC_FLAG_TG</i>	transmitted good frames more single collision flag

<i>FMSC</i>	
<i>ENET_MSC_FLAG_TG F</i>	transmitted good frames flag
<i>ENET_DMA_FLAG_TS</i>	transmit status flag
<i>ENET_DMA_FLAG_TP S</i>	transmit process stopped status flag
<i>ENET_DMA_FLAG_TB U</i>	transmit buffer unavailable status flag
<i>ENET_DMA_FLAG_TJ T</i>	transmit jabber timeout status flag
<i>ENET_DMA_FLAG_RO</i>	receive overflow status flag
<i>ENET_DMA_FLAG_TU</i>	transmit underflow status flag
<i>ENET_DMA_FLAG_RS</i>	receive status flag
<i>ENET_DMA_FLAG_RB U</i>	receive buffer unavailable status flag
<i>ENET_DMA_FLAG_RP S</i>	receive process stopped status flag
<i>ENET_DMA_FLAG_R WT</i>	receive watchdog timeout status flag
<i>ENET_DMA_FLAG_ET</i>	early transmit status flag
<i>ENET_DMA_FLAG_FB E</i>	fatal bus error status flag
<i>ENET_DMA_FLAG_ER</i>	early receive status flag
<i>ENET_DMA_FLAG_AI</i>	abnormal interrupt summary flag
<i>ENET_DMA_FLAG_NI</i>	normal interrupt summary flag
<i>ENET_DMA_FLAG_EB _DMA_ERROR</i>	DMA error flag
<i>ENET_DMA_FLAG_EB _TRANSFER_ERROR</i>	transfer error flag
<i>ENET_DMA_FLAG_EB _ACCESS_ERROR</i>	access error flag
<i>ENET_DMA_FLAG_MS C</i>	MSC status flag

<i>ENET_DMA_FLAG_WUM</i>	WUM status flag
<i>ENET_DMA_FLAG_TS_T</i>	timestamp trigger status flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check whether the specified flag bit is set */
```

```
enet_flag_get (ENET_DMA_FLAG_RS);
```

### enet\_flag\_clear

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

**Table 3-233. Function enet\_flag\_clear**

<b>Function name</b>	enet_flag_clear
<b>Function prototype</b>	void enet_flag_clear(enet_flag_clear_enum enet_flag);
<b>Function descriptions</b>	clear the ENET DMA status flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>enet_flag</b>	ENET DMA flag clear, refer to enet_flag_clear_enum only one parameter can be selected which is shown as below
<i>ENET_DMA_FLAG_TS_CLR</i>	transmit status flag clear
<i>ENET_DMA_FLAG_TPS_CLR</i>	transmit process stopped status flag clear
<i>ENET_DMA_FLAG_TBU_CLR</i>	transmit buffer unavailable status flag clear
<i>ENET_DMA_FLAG_TJT_CLR</i>	transmit jabber timeout status flag clear
<i>ENET_DMA_FLAG_RO</i>	receive overflow status flag clear

<code>_CLR</code>	
<code>ENET_DMA_FLAG_TU_CLR</code>	transmit underflow status flag clear
<code>ENET_DMA_FLAG_RS_CLR</code>	receive status flag clear
<code>ENET_DMA_FLAG_RBU_CLR</code>	receive buffer unavailable status flag clear
<code>ENET_DMA_FLAG_RPS_CLR</code>	receive process stopped status flag clear
<code>ENET_DMA_FLAG_RWT_CLR</code>	receive watchdog timeout status flag clear
<code>ENET_DMA_FLAG_ET_CLR</code>	early transmit status flag clear
<code>ENET_DMA_FLAG_FBE_CLR</code>	fatal bus error status flag clear
<code>ENET_DMA_FLAG_ER_CLR</code>	early receive status flag clear
<code>ENET_DMA_FLAG_AICLR</code>	abnormal interrupt summary flag clear
<code>ENET_DMA_FLAG_NICLR</code>	normal interrupt summary flag clear
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the specified flag bit */
```

```
enet_flag_clear(ENET_DMA_FLAG_RS_CLR);
```

### enet\_interrupt\_enable

The description of `enet_interrupt_enable` is shown as below:

**Table 3-234. Function `enet_interrupt_enable`**

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



<b>Function prototype</b>	void enet_interrupt_enable(enet_int_enum enet_int);
<b>Function descriptions</b>	enable ENET MAC/MSD/DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>enet_int</b>	ENET interrupt only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_WUMIM</i>	WUM interrupt mask
<i>ENET_MAC_INT_TMS TIM</i>	timestamp trigger interrupt mask
<i>ENET_MSC_INT_RFC EIM</i>	received frame CRC error interrupt mask
<i>ENET_MSC_INT_RFA EIM</i>	received frames alignment error interrupt mask
<i>ENET_MSC_INT_RGU FIM</i>	received good unicast frames interrupt mask
<i>ENET_MSC_INT_TGF SCIM</i>	transmitted good frames single collision interrupt mask
<i>ENET_MSC_INT_TGF MSCIM</i>	transmitted good frames more single collision interrupt mask
<i>ENET_MSC_INT_TGFI M</i>	transmitted good frames interrupt mask
<i>ENET_DMA_INT_TIE</i>	transmit interrupt enable
<i>ENET_DMA_INT_TPSI E</i>	transmit process stopped interrupt enable
<i>ENET_DMA_INT_TBUI E</i>	transmit buffer unavailable interrupt enable
<i>ENET_DMA_INT_TJTI E</i>	transmit jabber timeout interrupt enable
<i>ENET_DMA_INT_ROIE</i>	receive overflow interrupt enable
<i>ENET_DMA_INT_TUIE</i>	transmit underflow interrupt enable
<i>ENET_DMA_INT_RIE</i>	receive interrupt enable

<i>ENET_DMA_INT_RBUIE</i>	receive buffer unavailable interrupt enable
<i>ENET_DMA_INT_RPSIE</i>	receive process stopped interrupt enable
<i>ENET_DMA_INT_RWTIE</i>	receive watchdog timeout interrupt enable
<i>ENET_DMA_INT_ETIE</i>	early transmit interrupt enable
<i>ENET_DMA_INT_FBEIE</i>	fatal bus error interrupt enable
<i>ENET_DMA_INT_ERIE</i>	early receive interrupt enable
<i>ENET_DMA_INT_AIE</i>	abnormal interrupt summary enable
<i>ENET_DMA_INT_NIE</i>	normal interrupt summary enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable normal interrupt summary */
enet_interrupt_enable(ENET_DMA_INT_NIE);
```

### enet\_interrupt\_disable

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

**Table 3-235. Function enet\_interrupt\_disable**

<b>Function name</b>	enet_interrupt_disable
<b>Function prototype</b>	void enet_interrupt_disable(enet_int_enum enet_int);
<b>Function descriptions</b>	disable ENET MAC/MSD/DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>enet_int</b>	ENET interrupt only one parameter can be selected which is shown as below

<i>ENET_MAC_INT_WUMIM</i>	WUM interrupt mask
<i>ENET_MAC_INT_TMS TIM</i>	timestamp trigger interrupt mask
<i>ENET_MSC_INT_RFC EIM</i>	received frame CRC error interrupt mask
<i>ENET_MSC_INT_RFA EIM</i>	received frames alignment error interrupt mask
<i>ENET_MSC_INT_RGU FIM</i>	received good unicast frames interrupt mask
<i>ENET_MSC_INT_TGF SCIM</i>	transmitted good frames single collision interrupt mask
<i>ENET_MSC_INT_TGF MSCIM</i>	transmitted good frames more single collision interrupt mask
<i>ENET_MSC_INT_TGFI M</i>	transmitted good frames interrupt mask
<i>ENET_DMA_INT_TIE</i>	transmit interrupt enable
<i>ENET_DMA_INT_TPSI E</i>	transmit process stopped interrupt enable
<i>ENET_DMA_INT_TBUI E</i>	transmit buffer unavailable interrupt enable
<i>ENET_DMA_INT_TJTI E</i>	transmit jabber timeout interrupt enable
<i>ENET_DMA_INT_ROIE</i>	receive overflow interrupt enable
<i>ENET_DMA_INT_TUIE</i>	transmit underflow interrupt enable
<i>ENET_DMA_INT_RIE</i>	receive interrupt enable
<i>ENET_DMA_INT_RBUI E</i>	receive buffer unavailable interrupt enable
<i>ENET_DMA_INT_RPSI E</i>	receive process stopped interrupt enable
<i>ENET_DMA_INT_RWT IE</i>	receive watchdog timeout interrupt enable
<i>ENET_DMA_INT_ETIE</i>	early transmit interrupt enable
<i>ENET_DMA_INT_FBEI</i>	fatal bus error interrupt enable

<i>E</i>	
<i>ENET_DMA_INT_ERIE</i>	early receive interrupt enable
<i>ENET_DMA_INT_AIE</i>	abnormal interrupt summary enable
<i>ENET_DMA_INT_NIE</i>	normal interrupt summary enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable normal interrupt summary */
enet_interrupt_disable(ENET_DMA_INT_NIE);
```

### enet\_interrupt\_flag\_get

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

**Table 3-236. Function enet\_interrupt\_flag\_get**

<b>Function name</b>	enet_interrupt_flag_get
<b>Function prototype</b>	FlagStatus enet_interrupt_flag_get(enet_int_flag_enum int_flag);
<b>Function descriptions</b>	get ENET MAC/MSR/DMA interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	ENET interrupt flag only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_FLAG_WUM</i>	WUM status flag
<i>ENET_MAC_INT_FLAG_MSR</i>	MSR status flag
<i>ENET_MAC_INT_FLAG_MSCR</i>	MSR receive status flag
<i>ENET_MAC_INT_FLAG_MSCT</i>	MSR transmit status flag

<i>ENET_MAC_INT_FLG</i> <i>G_TMST</i>	time stamp trigger status flag
<i>ENET_MSC_INT_FLG</i> <i>G_RFCE</i>	received frames CRC error flag
<i>ENET_MSC_INT_FLG</i> <i>G_RFAE</i>	received frames alignment error flag
<i>ENET_MSC_INT_FLG</i> <i>G_RGUF</i>	received good unicast frames flag
<i>ENET_MSC_INT_FLG</i> <i>G_TGFSC</i>	transmitted good frames single collision flag
<i>ENET_MSC_INT_FLG</i> <i>G_TGFMSC</i>	transmitted good frames more single collision flag
<i>ENET_MSC_INT_FLG</i> <i>G_TGF</i>	transmitted good frames flag
<i>ENET_DMA_INT_FLG</i> <i>G_TS</i>	transmit status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TPS</i>	transmit process stopped status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TBU</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TJT</i>	transmit jabber timeout status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RO</i>	receive overflow status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TU</i>	transmit underflow status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RS</i>	receive status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RBU</i>	receive buffer unavailable status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RPS</i>	receive process stopped status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RWT</i>	receive watchdog timeout status flag
<i>ENET_DMA_INT_FLG</i>	early transmit status flag

<i>G_ET</i>	
<i>ENET_DMA_INT_FLG</i> <i>G_FBE</i>	fatal bus error status flag
<i>ENET_DMA_INT_FLG</i> <i>G_ER</i>	early receive status flag
<i>ENET_DMA_INT_FLG</i> <i>G_AI</i>	abnormal interrupt summary flag
<i>ENET_DMA_INT_FLG</i> <i>G_NI</i>	normal interrupt summary flag
<i>ENET_DMA_INT_FLG</i> <i>G_MSC</i>	MSC status flag
<i>ENET_DMA_INT_FLG</i> <i>G_WUM</i>	WUM status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TST</i>	timestamp trigger status flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

### enet\_interrupt\_flag\_clear

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

**Table 3-237. Function enet\_interrupt\_flag\_clear**

<b>Function name</b>	enet_interrupt_flag_clear
<b>Function prototype</b>	void enet_interrupt_flag_clear(enet_int_flag_clear_enum int_flag_clear);
<b>Function descriptions</b>	clear ENET DMA interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>int_flag_clear</b>	clear ENET interrupt flag only one parameter can be selected which is shown as below
<i>ENET_DMA_INT_FLG</i> <i>G_TS_CLR</i>	transmit status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TPS_CLR</i>	transmit process stopped status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TBU_CLR</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TJT_CLR</i>	transmit jabber timeout status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RO_CLR</i>	receive overflow status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TU_CLR</i>	transmit underflow status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RS_CLR</i>	receive status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RBU_CLR</i>	receive buffer unavailable status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RPS_CLR</i>	receive process stopped status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RWT_CLR</i>	receive watchdog timeout status flag
<i>ENET_DMA_INT_FLG</i> <i>G_ET_CLR</i>	early transmit status flag
<i>ENET_DMA_INT_FLG</i> <i>G_FBE_CLR</i>	fatal bus error status flag
<i>ENET_DMA_INT_FLG</i> <i>G_ER_CLR</i>	early receive status flag
<i>ENET_DMA_INT_FLG</i> <i>G_AI_CLR</i>	abnormal interrupt summary flag
<i>ENET_DMA_INT_FLG</i> <i>G_NI_CLR</i>	normal interrupt summary flag
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

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

```
enet_interrupt_flag_clear(ENET_DMA_INT_FLAG_RS);
```

### enet\_tx\_enable

The description of enet\_tx\_enable is shown as below:

**Table 3-238. Function enet\_tx\_enable**

<b>Function name</b>	enet_tx_enable
<b>Function prototype</b>	void enet_tx_enable(void);
<b>Function descriptions</b>	ENET Tx function enable (include MAC and DMA module)
<b>Precondition</b>	-
<b>The called functions</b>	enet_txfifo_flush()
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable transport function of MAC and DMA */
```

```
enet_tx_enable();
```

### enet\_tx\_disable

The description of enet\_tx\_disable is shown as below:

**Table 3-239. Function enet\_tx\_disable**

<b>Function name</b>	enet_tx_disable
<b>Function prototype</b>	void enet_tx_disable(void);
<b>Function descriptions</b>	ENET Tx function disable (include MAC and DMA module)



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

Example:

```
/* disable transport function of MAC and DMA */
```

```
enet_tx_disable();
```

### enet\_rx\_enable

The description of enet\_rx\_enable is shown as below:

**Table 3-240. Function enet\_rx\_enable**

<b>Function name</b>	enet_rx_enable
<b>Function prototype</b>	void enet_rx_enable(void);
<b>Function descriptions</b>	ENET Rx function enable (include MAC and DMA module)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable reception function of MAC and DMA */
```

```
enet_rx_enable();
```

## enet\_rx\_disable

The description of enet\_rx\_disable is shown as below:

**Table 3-241. Function enet\_rx\_disable**

<b>Function name</b>	enet_rx_disable
<b>Function prototype</b>	void enet_rx_disable(void);
<b>Function descriptions</b>	ENET Rx function disable (include MAC and DMA module)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable reception function of MAC and DMA */
```

```
enet_rx_disable();
```

## enet\_registers\_get

The description of enet\_registers\_get is shown as below:

**Table 3-242. Function enet\_registers\_get**

<b>Function name</b>	enet_registers_get
<b>Function prototype</b>	void enet_registers_get(enet_registers_type_enum type, uint32_t *preg, uint32_t num);
<b>Function descriptions</b>	put registers value into the application buffer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>type</b>	register type which will be get only one parameter can be selected which is shown as below

<i>ALL_MAC_REG</i>	get the registers within the offset scope range ENET_MAC_CFG to ENET_MAC_FCTH
<i>ALL_MSC_REG</i>	get the registers within the offset scope range ENET_MSC_CTL to ENET_MSC_RGUFCNT
<i>ALL_PTP_REG</i>	get the registers within the offset scope range ENET_PTP_TSCTL to ENET_PTP_PPSCTL
<i>ALL_DMA_REG</i>	get the registers within the offset scope range ENET_DMA_BCTL to ENET_DMA_CRBADDR
<b>Input parameter{in}</b>	
<b>num</b>	the number of registers that the user want to get, (0 -- 54)
<b>Output parameter{out}</b>	
<b>preg</b>	the application buffer pointer for storing the register value
<b>Return value</b>	
-	-

Example:

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

```
uint32_t register_buffer[5];
```

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

### enet\_debug\_status\_get

The description of enet\_debug\_status\_get is shown as below:

**Table 3-243. Function enet\_debug\_status\_get**

<b>Function name</b>	enet_debug_status_get
<b>Function prototype</b>	uint32_t enet_debug_status_get(uint32_t mac_debug);
<b>Function descriptions</b>	get the enet debug status from the debug register
<b>Precondition</b>	-
<b>The called functions</b>	--
<b>Input parameter{in}</b>	
<b>mac_debug</b>	enet debug status only one parameter can be selected which is shown as below
<i>ENET_MAC_RECEIVE</i>	MAC receiver is not in idle state

<i>R_NOT_IDLE</i>	
<i>ENET_RX_ASYNC RONOUS_FIFO_STATE</i>	Rx asynchronous FIFO status
<i>ENET_RXFIFO_WRITE NG</i>	RxFIFO is doing write operation
<i>ENET_RXFIFO_READ _STATUS</i>	RxFIFO read operation status
<i>ENET_RXFIFO_STATE</i>	RxFIFO state
<i>ENET_MAC_TRANSMI TTER_NOT_IDLE</i>	MAC transmitter is not in idle state
<i>ENET_MAC_TRANSMI TTER_STATUS</i>	status of MAC transmitter
<i>ENET_PAUSE_CONDI TION_STATUS</i>	pause condition status
<i>ENET_TXFIFO_READ _STATUS</i>	TxFIFO read operation status
<i>ENET_TXFIFO_WRITE NG</i>	TxFIFO is doing write operation
<i>ENET_TXFIFO_NOT_E MPTY</i>	TxFIFO is not empty
<i>ENET_TXFIFO_FULL</i>	TxFIFO is full
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	value of the status users want to get

Example:

```
/* get debug message of RxFIFO state */
```

```
uint32_t debug_value;
```

```
debug_value = enet_debug_status_get (ENET_RXFIFO_STATE);
```

### **enet\_address\_filter\_enable**

The description of enet\_address\_filter\_enable is shown as below:

Table 3-244. Function `enet_address_filter_enable`

Function name	<code>enet_address_filter_enable</code>
Function prototype	<code>void enet_address_filter_enable(enet_macaddress_enum mac_addr);</code>
Function descriptions	enable the MAC address filter
Precondition	-
The called functions	--
Input parameter{in}	
<b>mac_addr</b>	select which MAC address will be enable only one parameter can be selected which is shown as below
<code>ENET_MAC_ADDRES S1</code>	enable MAC address 1 filter
<code>ENET_MAC_ADDRES S2</code>	enable MAC address 2 filter
<code>ENET_MAC_ADDRES S3</code>	enable MAC address 3 filter
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_address_filter_enable(ENET_MAC_ADDRESS1);
```

### **enet\_address\_filter\_disable**

The description of `enet_address_filter_disable` is shown as below:

Table 3-245. Function `enet_address_filter_disable`

Function name	<code>enet_address_filter_disable</code>
Function prototype	<code>void enet_address_filter_disable(enet_macaddress_enum mac_addr);</code>
Function descriptions	disable the MAC address filter
Precondition	-
The called functions	-

Input parameter{in}	
<b>mac_addr</b>	select which MAC address will be enable only one parameter can be selected which is shown as below
<i>ENET_MAC_ADDRES</i> <i>S1</i>	enable MAC address 1 filter
<i>ENET_MAC_ADDRES</i> <i>S2</i>	enable MAC address 2 filter
<i>ENET_MAC_ADDRES</i> <i>S3</i>	enable MAC address 3 filter
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_address_filter_disable(ENET_MAC_ADDRESS1);
```

### enet\_address\_filter\_config

The description of enet\_address\_filter\_config is shown as below:

**Table 3-246. Function enet\_address\_filter\_config**

<b>Function name</b>	enet_address_filter_config
<b>Function prototype</b>	void enet_address_filter_config(enet_macaddress_enum mac_addr, uint32_t addr_mask, uint32_t filter_type);
<b>Function descriptions</b>	configure the MAC address filter
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>mac_addr</b>	select which MAC address will be enable only one parameter can be selected which is shown as below
<i>ENET_MAC_ADDRES</i> <i>S1</i>	enable MAC address 1 filter
<i>ENET_MAC_ADDRES</i>	enable MAC address 2 filter

S2	
ENET_MAC_ADDRES S3	enable MAC address 3 filter
<b>Input parameter{in}</b>	
<b>addr_mask</b>	select which MAC address bytes will be mask one or more parameters can be selected which are shown as below
ENET_ADDRESS_MA SK_BYTE0	mask ENET_MAC_ADDR1L[7:0] bits
ENET_ADDRESS_MA SK_BYTE1	mask ENET_MAC_ADDR1L[15:8] bits
ENET_ADDRESS_MA SK_BYTE2	mask ENET_MAC_ADDR1L[23:16] bits
ENET_ADDRESS_MA SK_BYTE3	mask ENET_MAC_ADDR1L [31:24] bits
ENET_ADDRESS_MA SK_BYTE4	mask ENET_MAC_ADDR1H [7:0] bits
ENET_ADDRESS_MA SK_BYTE5	mask ENET_MAC_ADDR1H [15:8] bits
<b>Input parameter{in}</b>	
<b>filter_type</b>	select which MAC address filter type will be selected only one parameter can be selected which is shown as below
ENET_ADDRESS_FILT ER_SA	The MAC address is used to compared with the SA field of the received frame
ENET_ADDRESS_FILT ER_DA	The MAC address is used to compared with the DA field of the received frame
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

## enet\_phy\_config

The description of enet\_phy\_config is shown as below:

**Table 3-247. Function enet\_phy\_config**

<b>Function name</b>	enet_phy_config
<b>Function prototype</b>	ErrStatus enet_phy_config(void);
<b>Function descriptions</b>	PHY interface configuration (configure SMI clock and reset PHY chip)
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get()/enet_phy_write_read()
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* config PHY interface */
```

```
enet_phy_config();
```

## enet\_phy\_write\_read

The description of enet\_phy\_write\_read is shown as below:

**Table 3-248. Function enet\_phy\_write\_read**

<b>Function name</b>	enet_phy_write_read
<b>Function prototype</b>	ErrStatus enet_phy_write_read(enet_phydirection_enum direction, uint16_t phy_address, uint16_t phy_reg, uint16_t *pvalue);
<b>Function descriptions</b>	write to / read from a PHY register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>direction</b>	only one parameter can be selected which is shown as below
<i>ENET_PHY_WRITE</i>	write data to phy register



<b>ENET_PHY_READ</b>	read data from phy register
<b>Input parameter{in}</b>	
<b>phy_address</b>	0x0 - 0x1F
<b>Input parameter{in}</b>	
<b>phy_reg</b>	0x0 - 0x1F
<b>Input parameter{in}</b>	
<b>pvalue</b>	the value will be written to the PHY register in ENET_PHY_WRITE direction
<b>Output parameter{out}</b>	
<b>pvalue</b>	the value will be read from the PHY register in ENET_PHY_READ direction
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* write 0 to PHY BCR register */
```

```
uint16_t temp_phy = 0U;
```

```
phy_state = enet_phy_write_read(ENET_PHY_WRITE, PHY_ADDRESS, PHY_REG_BCR,
&temp_phy);
```

### enet\_phyloopback\_enable

The description of enet\_phyloopback\_enable is shown as below:

**Table 3-249. Function enet\_phyloopback\_enable**

<b>Function name</b>	enet_phyloopback_enable
<b>Function prototype</b>	ErrStatus enet_phyloopback_enable(void);
<b>Function descriptions</b>	enable the loopback function of PHY chip
<b>Precondition</b>	-
<b>The called functions</b>	enet_phy_write_read()
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

<b>ErrStatus</b>	ERROR or SUCCESS
------------------	------------------

Example:

```
/* enable the loopback function of PHY chip */
```

```
ErrStatus phy_state = ERROR;
```

```
phy_state = enet_phyloopback_enable();
```

### enet\_phyloopback\_disable

The description of enet\_phyloopback\_disable is shown as below:

**Table 3-250. Function enet\_phyloopback\_disable**

<b>Function name</b>	enet_phyloopback_disable
<b>Function prototype</b>	ErrStatus enet_phyloopback_disable(void);
<b>Function descriptions</b>	disable the loopback function of PHY chip
<b>Precondition</b>	-
<b>The called functions</b>	enet_phy_write_read
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* disable the loopback function of PHY chip */
```

```
ErrStatus phy_state = ERROR;
```

```
phy_state = enet_phyloopback_disable();
```

### enet\_forward\_feature\_enable

The description of enet\_forward\_feature\_enable is shown as below:

**Table 3-251. Function enet\_forward\_feature\_enable**

<b>Function name</b>	enet_forward_feature_enable
<b>Function prototype</b>	void enet_forward_feature_enable(uint32_t feature);

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

Example:

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

### enet\_forward\_feature\_disable

The description of enet\_forward\_feature\_disable is shown as below:

**Table 3-252. Function enet\_forward\_feature\_disable**

<b>Function name</b>	enet_forward_feature_disable
<b>Function prototype</b>	void enet_forward_feature_disable(uint32_t feature);
<b>Function descriptions</b>	disable ENET forward feature
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>feature</b>	the feature of ENET forward mode one or more parameters can be selected which are shown as below
<i>ENET_AUTO_PADCRC_DROP</i>	the function of the MAC strips the Pad/FCS field on received frames
<i>ENET_TYPEFRAME_CRC_DROP</i>	the function that FCS field(last 4 bytes) of frame will be dropped before forwarding
<i>ENET_FORWARD_ERROR_FRAMES</i>	the function that all frame received with error except runt error are forwarded to memory
<i>ENET_FORWARD_UNDERSZ_GOODFRAMES</i>	the function that forwarding undersized good frames
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### enet\_fliter\_feature\_enable

The description of enet\_fliter\_feature\_enable is shown as below:

**Table 3-253. Function enet\_fliter\_feature\_enable**

<b>Function name</b>	enet_fliter_feature_enable
<b>Function prototype</b>	void enet_fliter_feature_enable(uint32_t feature);
<b>Function descriptions</b>	enable ENET fliter feature
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>feature</b>	the feature of ENET fliter mode one or more parameters can be selected which are shown as below

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

Example:

```
/* enable filter source address function */
```

```
enet_fliter_feature_enable(ENET_SRC_FILTER);
```

### **enet\_fliter\_feature\_disable**

The description of `enet_fliter_feature_disable` is shown as below:

**Table 3-254. Function `enet_fliter_feature_disable`**

<b>Function name</b>	<code>enet_fliter_feature_disable</code>
<b>Function prototype</b>	<code>void enet_fliter_feature_disable(uint32_t feature);</code>
<b>Function descriptions</b>	disable ENET fliter feature
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>feature</b>	the feature of ENET fliter mode

	one or more parameters can be selected which are shown as below
<i>ENET_SRC_FILTER</i>	filter source address function
<i>ENET_SRC_FILTER_INVERSE</i>	inverse source address filtering result function
<i>ENET_DEST_FILTER_INVERSE</i>	inverse DA filtering result function
<i>ENET_MULTICAST_FILTER_PASS</i>	pass all multicast frames function
<i>ENET_MULTICAST_FILTER_HASH_MODE</i>	HASH multicast filter function
<i>ENET_UNICAST_FILTER_HASH_MODE</i>	HASH unicast filter function
<i>ENET_FILTER_MODE_EITHER</i>	HASH or perfect filter function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable filter source address function */
```

```
enet_fliter_feature_enable(ENET_SRC_FILTER);
```

### enet\_pauseframe\_generate

The description of enet\_pauseframe\_generate is shown as below:

**Table 3-255. Function enet\_pauseframe\_generate**

<b>Function name</b>	enet_pauseframe_generate
<b>Function prototype</b>	ErrStatus enet_pauseframe_generate(void);
<b>Function descriptions</b>	generate the pause frame, ENET will send pause frame after enable transmit flow control this function only use in full-duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* generate the pause frame */
```

```
ErrStatus reval;
```

```
reval = enet_pauseframe_generate();
```

### enet\_pauseframe\_detect\_config

The description of enet\_pauseframe\_detect\_config is shown as below:

**Table 3-256. Function enet\_pauseframe\_detect\_config**

<b>Function name</b>	enet_pauseframe_detect_config
<b>Function prototype</b>	void enet_pauseframe_detect_config(uint32_t detect);
<b>Function descriptions</b>	configure the pause frame detect type
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>detect</b>	pause frame detect type only one parameter can be selected which is shown as below
ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSE_DETECT	besides the unique multicast address, MAC can also use the MAC0 address to detecting pause frame
ENET_UNIQUE_PAUSE_DETECT	only the unique multicast address for pause frame which is specified in IEEE802.3 can be detected
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config the pause frame detect type as ENET_UNIQUE_PAUSEDDETECT */
```

```
enet_pauseframe_detect_config(ENET_UNIQUE_PAUSEDDETECT);
```

### enet\_pauseframe\_config

The description of enet\_pauseframe\_config is shown as below:

**Table 3-257. Function enet\_pauseframe\_config**

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

Example:

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



```
enet_pauseframe_config(30, ENET_PAUSETIME_MINUS4);
```

### enet\_flowcontrol\_threshold\_config

The description of enet\_flowcontrol\_threshold\_config is shown as below:

**Table 3-258. Function enet\_flowcontrol\_threshold\_config**

Function name	enet_flowcontrol_threshold_config
Function prototype	void enet_flowcontrol_threshold_config(uint32_t deactive, uint32_t active);
Function descriptions	configure the threshold of the flow control(deactive and active threshold)
Precondition	-
The called functions	-
Input parameter{in}	
<b>deactive</b>	the threshold of the deactive flow control, this value should always be less than active flow control value, only one parameter can be selected which is shown as below
ENET_DEACTIVE_TH RESHOLD_256BYTES	threshold level is 256 bytes
ENET_DEACTIVE_TH RESHOLD_512BYTES	threshold level is 512 bytes
ENET_DEACTIVE_TH RESHOLD_768BYTES	threshold level is 768 bytes
ENET_DEACTIVE_TH RESHOLD_1024BYTE S	threshold level is 1024 bytes
ENET_DEACTIVE_TH RESHOLD_1280BYTE S	threshold level is 1280 bytes
ENET_DEACTIVE_TH RESHOLD_1536BYTE S	threshold level is 1536 bytes
ENET_DEACTIVE_TH RESHOLD_1792BYTE S	threshold level is 1792 bytes
Input parameter{in}	
<b>active</b>	the threshold of the active flow control, only one parameter can be selected which is shown as below

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

Example:

```
/* configure the threshold of the flow control */
```

```
enet_flowcontrol_threshold_config(ENET_DEACTIVE_THRESHOLD_256BYTES, ENET_A  
CTIVE_THRESHOLD_256BYTES);
```

### **enet\_flowcontrol\_feature\_enable**

The description of enet\_flowcontrol\_feature\_enable is shown as below:

**Table 3-259. Function enet\_flowcontrol\_feature\_enable**

<b>Function name</b>	enet_flowcontrol_feature_enable
<b>Function prototype</b>	void enet_flowcontrol_feature_enable(uint32_t feature);
<b>Function descriptions</b>	enable ENET flow control feature
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>feature</b>	the feature of ENET flow control mode one or more parameters can be selected which are shown as below
<i>ENET_ZERO_QUANTA_PAUSE</i>	the automatic zero-quanta generation function
<i>ENET_TX_FLOWCONTROL</i>	the flow control operation in the MAC
<i>ENET_RX_FLOWCONTROL</i>	decoding function for the received pause frame and process it
<i>ENET_BACK_PRESSURE</i>	back pressure operation in the MAC(only use in half-duplex mode)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the flow control operation in the MAC */
```

```
enet_flowcontrol_feature_enable(ENET_ZERO_QUANTA_PAUSE);
```

### **enet\_flowcontrol\_feature\_disable**

The description of enet\_flowcontrol\_feature\_disable is shown as below:

**Table 3-260. Function enet\_flowcontrol\_feature\_disable**

<b>Function name</b>	enet_flowcontrol_feature_disable
<b>Function prototype</b>	void enet_flowcontrol_feature_disable(uint32_t feature);
<b>Function descriptions</b>	disable ENET flow control feature
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>feature</b>	the feature of ENET flow control mode one or more parameters can be selected which are shown as below
<i>ENET_ZERO_QUANTA_PAUSE</i>	the automatic zero-quanta generation function

<i>ENET_TX_FLOWCONTROL</i>	the flow control operation in the MAC
<i>ENET_RX_FLOWCONTROL</i>	decoding function for the received pause frame and process it
<i>ENET_BACK_PRESSURE</i>	back pressure operation in the MAC(only use in half-duplex mode)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the automatic zero-quantum generation function */
```

```
enet_flowcontrol_feature_disable(ENET_ZERO_QUANTA_PAUSE);
```

### enet\_dmaprocess\_state\_get

The description of enet\_dmaprocess\_state\_get is shown as below:

**Table 3-261. Function enet\_dmaprocess\_state\_get**

<b>Function name</b>	enet_dmaprocess_state_get
<b>Function prototype</b>	uint32_t enet_dmaprocess_state_get(enet_dmadirection_enum direction);
<b>Function descriptions</b>	get the dma transmit/receive process state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>direction</b>	choose the direction of DMA process which users want to resume only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA transmit process
<i>ENET_DMA_RX</i>	DMA receive process
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

<b>uint32_t</b>	<p>state of dma process, the value range shows below:</p> <p>ENET_RX_STATE_STOPPED / ENET_RX_STATE_FETCHING /  ENET_RX_STATE_WAITING / ENET_RX_STATE_SUSPENDED /  ENET_RX_STATE_CLOSING / ENET_RX_STATE_QUEUING /  ENET_TX_STATE_STOPPED / ENET_TX_STATE_FETCHING /  ENET_TX_STATE_WAITING / ENET_TX_STATE_READING /  ENET_TX_STATE_SUSPENDED / ENET_TX_STATE_CLOSING</p>
-----------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Example:

```

/* get the dma receive process state */

uint32_t reval;

reval = enet_dmaprocess_state_get(ENET_DMA_RX);

if(ENET_RX_STATE_SUSPENDED == reval){

    do...

}

```

### enet\_dmaprocess\_resume

The description of enet\_dmaprocess\_resume is shown as below:

**Table 3-262. Function enet\_dmaprocess\_resume**

<b>Function name</b>	enet_dmaprocess_resume
<b>Function prototype</b>	void enet_dmaprocess_resume(enet_dmadirection_enum direction);
<b>Function descriptions</b>	poll the DMA transmission/reception enable by writing any value to the ENET_DMA_TPEN/ENET_DMA_RPEN register, this will make the DMA to resume transmission/reception
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>direction</b>	choose the direction of DMA process which users want to resume only one parameter can be selected which is shown as below
ENET_DMA_TX	DMA transmit process
ENET_DMA_RX	DMA receive process
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* enable DMA receive process */
```

```
enet_dmaprocess_resume(ENET_DMA_RX);
```

### enet\_rxprocess\_check\_recovery

The description of enet\_rxprocess\_check\_recovery is shown as below:

**Table 3-263. Function enet\_rxprocess\_check\_recovery**

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

Example:

```
/* check and recover the Rx process */
```

```
enet_rxprocess_check_recovery();
```

### enet\_txfifo\_flush

The description of enet\_txfifo\_flush is shown as below:

**Table 3-264. Function enet\_txfifo\_flush**

<b>Function name</b>	enet_txfifo_flush
<b>Function prototype</b>	ErrStatus enet_txfifo_flush(void);
<b>Function descriptions</b>	flush the ENET transmit FIFO, and wait until the flush operation completes

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

Example:

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

```
ErrStatus reval = ERROR;
```

```
reval = enet_txfifo_flush();
```

### enet\_current\_desc\_address\_get

The description of enet\_current\_desc\_address\_get is shown as below:

**Table 3-265. Function enet\_current\_desc\_address\_get**

<b>Function name</b>	enet_current_desc_address_get
<b>Function prototype</b>	uint32_t enet_current_desc_address_get(enet_desc_reg_enum addr_get);
<b>Function descriptions</b>	get the transmit/receive address of current descriptor, or current buffer, or descriptor table
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>addr_get</b>	choose the address which users want to get only one parameter can be selected which is shown as below
<i>ENET_RX_DESC_TABLE</i>	the start address of the receive descriptor table
<i>ENET_RX_CURRENT_DESC</i>	the start descriptor address of the current receive descriptor read by the RxDMA controller
<i>ENET_RX_CURRENT_BUFFER</i>	the current receive buffer address being read by the RxDMA controller

<i>ENET_TX_DESC_TABLE</i>	the start address of the transmit descriptor table
<i>ENET_TX_CURRENT_DESC</i>	the start descriptor address of the current transmit descriptor read by the TxDMA controller
<i>ENET_TX_CURRENT_BUFFER</i>	the current transmit buffer address being read by the TxDMA controller
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	0- 0xFFFFFFFF

Example:

```
/* get the start address of the receive descriptor table */
```

```
uint32_t reval;
```

```
reval = enet_current_desc_address_get(ENET_RX_DESC_TABLE);
```

### enet\_desc\_information\_get

The description of enet\_desc\_information\_get is shown as below:

**Table 3-266. Function enet\_desc\_information\_get**

<b>Function name</b>	enet_desc_information_get
<b>Function prototype</b>	uint32_t enet_desc_information_get(enet_descriptors_struct *desc, enet_descstate_enum info_get);
<b>Function descriptions</b>	get the Tx or Rx descriptor information
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>desc</b>	the descriptor pointer which users want to get information, the structure members can refer to <a href="#">Structure enet_descriptors_struct</a>
<b>Input parameter{in}</b>	
<b>info_get</b>	the descriptor information type which is selected only one parameter can be selected which is shown as below
<i>RXDESC_BUFFER_1_SIZE</i>	receive buffer 1 size



<i>RXDESC_BUFFER_2_SIZE</i>	receive buffer 2 size
<i>RXDESC_FRAME_LENGTH</i>	the byte length of the received frame that was transferred to the buffer
<i>TXDESC_COLLISION_COUNT</i>	the number of collisions occurred before the frame was transmitted
<i>RXDESC_BUFFER_1_ADDR</i>	the buffer1 address of the Rx frame
<i>TXDESC_BUFFER_1_ADDR</i>	the buffer1 address of the Tx frame
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	descriptor information if value is 0xFFFFFFFFU, means the false input parameter

Example:

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

```
uint32_t reval;
```

```
reval = enet_desc_information_get(rx_desc, RXDESC_BUFFER_1_SIZE);
```

### enet\_missed\_frame\_counter\_get

The description of enet\_missed\_frame\_counter\_get is shown as below:

**Table 3-267. Function enet\_missed\_frame\_counter\_get**

<b>Function name</b>	enet_missed_frame_counter_get
<b>Function prototype</b>	void enet_missed_frame_counter_get(uint32_t *rxfifo_drop, uint32_t *rxdma_drop);
<b>Function descriptions</b>	get the number of missed frames during receiving
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	

<b>rxfifo_drop</b>	pointer to the number of frames dropped by RxFIFO
<b>Output parameter{out}</b>	
<b>rxdma_drop</b>	pointer to the number of frames missed by the RxDMA controller
<b>Return value</b>	
-	-

Example:

```
/* get the number of missed frames during receiving */
```

```
uint32_t rxcnt, txcnt;
```

```
enet_missed_frame_counter_get(&rxcnt, &txcnt);
```

### enet\_desc\_flag\_get

The description of enet\_desc\_flag\_get is shown as below:

**Table 3-268. Function enet\_desc\_flag\_get**

<b>Function name</b>	enet_desc_flag_get
<b>Function prototype</b>	FlagStatus enet_desc_flag_get(enet_descriptors_struct *desc, uint32_t desc_flag);
<b>Function descriptions</b>	get the bit flag of ENET DMA descriptor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>desc</b>	the descriptor pointer which users want to get flag, the structure members can refer to <a href="#">Structure enet_descriptors_struct</a>
<b>Input parameter{in}</b>	
<b>desc_flag</b> (the value according to the parameter <b>desc</b> )	the bit flag of ENET DMA descriptor only one parameter can be selected which is shown as below
When type of parameter <b>desc</b> is TX	
<i>ENET_TDES0_DB</i>	deferred
<i>ENET_TDES0_UFE</i>	underflow error
<i>ENET_TDES0_EXD</i>	excessive deferral
<i>ENET_TDES0_VFRM</i>	VLAN frame

<i>ENET_TDES0_ECO</i>	excessive collision
<i>ENET_TDES0_LCO</i>	late collision
<i>ENET_TDES0_NCA</i>	no carrier
<i>ENET_TDES0_LCA</i>	loss of carrier
<i>ENET_TDES0_IPPE</i>	IP payload error
<i>ENET_TDES0_FRMF</i>	frame flushed
<i>ENET_TDES0_JT</i>	jabber timeout
<i>ENET_TDES0_ES</i>	error summary
<i>ENET_TDES0_IPHE</i>	IP header error
<i>ENET_TDES0_TTMSS</i>	transmit timestamp status
<i>ENET_TDES0_TCHM</i>	the second address chained mode
<i>ENET_TDES0_TERM</i>	transmit end of ring mode
<i>ENET_TDES0_TTSEN</i>	transmit timestamp function enable
<i>ENET_TDES0_DPAD</i>	disable adding pad
<i>ENET_TDES0_DCRC</i>	disable CRC
<i>ENET_TDES0_FSG</i>	first segment
<i>ENET_TDES0_LSG</i>	last segment
<i>ENET_TDES0_INTC</i>	interrupt on completion
<i>ENET_TDES0_DAV</i>	DAV bit
When type of parameter <b>desc</b> is RX	
<i>ENET_RDES0_PCERR</i>	payload checksum error
<i>ENET_RDES0_CERR</i>	CRC error
<i>ENET_RDES0_DBERR</i>	dribble bit error
<i>ENET_RDES0_RERR</i>	receive error
<i>ENET_RDES0_RWDT</i>	receive watchdog timeout
<i>ENET_RDES0_FRMT</i>	frame type
<i>ENET_RDES0_LCO</i>	late collision
<i>ENET_RDES0_IPHER</i> <i>R</i>	IP frame header error

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

Example:

```
/* get the bit flag of ENET DMA descriptor */
```

```
FlagStatus reval;
```

```
reval = enet_desc_flag_get(p_txdesc, ENET_TDES0_TCHM);
```

### enet\_desc\_flag\_set

The description of enet\_desc\_flag\_set is shown as below:

**Table 3-269. Function enet\_desc\_flag\_set**

<b>Function name</b>	enet_desc_flag_set
<b>Function prototype</b>	void enet_desc_flag_set(enet_descriptors_struct *desc, uint32_t desc_flag);
<b>Function descriptions</b>	set the bit flag of ENET DMA descriptor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>desc</b>	the descriptor pointer which users want to set flag, the structure members can refer to <a href="#">Structure enet_descriptors_struct</a>

Input parameter{in}	
<b>desc_flag</b> (the value according to the parameter <b>desc</b> )	the bit flag of ENET DMA descriptor only one parameter can be selected which is shown as below
When type of parameter <b>desc</b> is TX	
<i>ENET_TDES0_VFRM</i>	VLAN frame
<i>ENET_TDES0_FRMF</i>	frame flushed
<i>ENET_TDES0_TCHM</i>	the second address chained mode
<i>ENET_TDES0_TERM</i>	transmit end of ring mode
<i>ENET_TDES0_TTSEN</i>	transmit timestamp function enable
<i>ENET_TDES0_DPAD</i>	disable adding pad
<i>ENET_TDES0_DCRC</i>	disable CRC
<i>ENET_TDES0_FSG</i>	first segment
<i>ENET_TDES0_LSG</i>	last segment
<i>ENET_TDES0_INTC</i>	interrupt on completion
<i>ENET_TDES0_DAV</i>	DAV bit
When type of parameter <b>desc</b> is RX	
<i>ENET_RDES0_DAV</i>	descriptor available
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### enet\_desc\_flag\_clear

The description of enet\_desc\_flag\_clear is shown as below:

**Table 3-270. Function enet\_desc\_flag\_clear**

<b>Function name</b>	enet_desc_flag_clear
----------------------	----------------------

<b>Function prototype</b>	void enet_desc_flag_clear(enet_descriptors_struct *desc, uint32_t desc_flag);
<b>Function descriptions</b>	clear the bit flag of ENET DMA descriptor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>desc</b>	the descriptor pointer which users want to set flag, the structure members can refer to <a href="#">Structure enet_descriptors_struct</a>
<b>Input parameter{in}</b>	
<b>desc_flag</b> (the value according to the parameter <b>desc</b> )	the bit flag of ENET DMA descriptor only one parameter can be selected which is shown as below
When type of parameter <b>desc</b> is TX	
<i>ENET_TDES0_VFRM</i>	VLAN frame
<i>ENET_TDES0_FRMF</i>	frame flushed
<i>ENET_TDES0_TCHM</i>	the second address chained mode
<i>ENET_TDES0_TERM</i>	transmit end of ring mode
<i>ENET_TDES0_TTSEN</i>	transmit timestamp function enable
<i>ENET_TDES0_DPAD</i>	disable adding pad
<i>ENET_TDES0_DCRC</i>	disable CRC
<i>ENET_TDES0_FSG</i>	first segment
<i>ENET_TDES0_LSG</i>	last segment
<i>ENET_TDES0_INTC</i>	interrupt on completion
<i>ENET_TDES0_DAV</i>	DAV bit
When type of parameter <b>desc</b> is RX	
<i>ENET_RDES0_DAV</i>	descriptor available
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### enet\_rx\_desc\_immediate\_receive\_complete\_interrupt

The description of enet\_rx\_desc\_immediate\_receive\_complete\_interrupt is shown as below:

**Table 3-271. Function enet\_rx\_desc\_immediate\_receive\_complete\_interrupt**

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

Example:

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

### enet\_rx\_desc\_delay\_receive\_complete\_interrupt

The description of enet\_rx\_desc\_delay\_receive\_complete\_interrupt is shown as below:

**Table 3-272. Function enet\_rx\_desc\_delay\_receive\_complete\_interrupt**

<b>Function name</b>	enet_rx_desc_delay_receive_complete_interrupt
<b>Function prototype</b>	void enet_rx_desc_delay_receive_complete_interrupt(enet_descriptors_struct *desc, uint32_t delay_time);

<b>Function descriptions</b>	when receiving completed, set RS bit in ENET_DMA_STAT register will is set after a configurable delay time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>desc</b>	the descriptor pointer which users want to get flag, the structure members can refer to <a href="#">Structure enet_descriptors_struct</a>
<b>Input parameter{in}</b>	
<b>delay_time</b>	delay a time of 256*delay_time HCLK(0x00000000 - 0x000000FF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### enet\_rxframe\_drop

The description of enet\_rxframe\_drop is shown as below:

**Table 3-273. Function enet\_rxframe\_drop**

<b>Function name</b>	enet_rxframe_drop
<b>Function prototype</b>	void enet_rxframe_drop(void);
<b>Function descriptions</b>	drop current receive frame
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-



Return value	
-	-

Example:

```
/* drop current receive frame */
```

```
enet_rxframe_drop( );
```

### enet\_dma\_feature\_enable

The description of enet\_dma\_feature\_enable is shown as below:

**Table 3-274. Function enet\_dma\_feature\_enable**

Function name	enet_dma_feature_enable
Function prototype	void enet_dma_feature_enable(uint32_t feature);
Function descriptions	enable DMA feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of DMA mode one or more parameters can be selected which are shown as below
ENET_NO_FLUSH_RX FRAME	RxDMA does not flushes frames function
ENET_SECONDFRAM E_OPT	TxDMA controller operate on second frame function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable RxDMA does not flushes frames function */
```

```
enet_dma_feature_enable(ENET_NO_FLUSH_RXFRAME);
```

### enet\_dma\_feature\_disable

The description of enet\_dma\_feature\_disable is shown as below:

Table 3-275. Function `enet_dma_feature_disable`

Function name	<code>enet_dma_feature_disable</code>
Function prototype	<code>void enet_dma_feature_disable(uint32_t feature);</code>
Function descriptions	disable DMA feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of DMA mode one or more parameters can be selected which are shown as below
<code>ENET_NO_FLUSH_RXFRAME</code>	RxDMA does not flushes frames function
<code>ENET_SECONDFRAME_E_OPT</code>	TxDMA controller operate on second frame function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RxDMA does not flushes frames function */
```

```
enet_dma_feature_disable(ENET_NO_FLUSH_RXFRAME);
```

### `enet_rx_desc_enhanced_status_get`

The description of `enet_rx_desc_enhanced_status_get` is shown as below:

Table 3-276. Function `enet_rx_desc_enhanced_status_get`

Function name	<code>enet_rx_desc_enhanced_status_get</code>
Function prototype	<code>uint32_t enet_rx_desc_enhanced_status_get(enet_descriptors_struct *desc, uint32_t desc_status);</code>
Function descriptions	get the bit of extended status flag in ENET DMA descriptor
Precondition	-
The called functions	-
Input parameter{in}	

<b>desc</b>	the descriptor pointer which users want to get the extended status flag, the structure members can refer to <a href="#">Structure enet descriptors struct</a>
<b>Input parameter{in}</b>	
<b>desc_status</b>	desc_status: the extended status want to get only one parameter can be selected which is shown as below
<i>ENET_RDES4_IPPLDT</i>	IP frame payload type
<i>ENET_RDES4_IPHER R</i>	IP frame header error
<i>ENET_RDES4_IPPLDE RR</i>	IP frame payload error
<i>ENET_RDES4_IPCKS B</i>	IP frame checksum bypassed
<i>ENET_RDES4_IPF4</i>	IP frame in version 4
<i>ENET_RDES4_IPF6</i>	IP frame in version 6
<i>ENET_RDES4_PTPMT</i>	PTP message type
<i>ENET_RDES4_PTPOE F</i>	PTP on ethernet frame
<i>ENET_RDES4_PTPVF</i>	PTP version format
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* get the IP frame payload type in ENET DMA descriptor */
```

```
uint32_t status;
```

```
status = enet_rx_desc_enhanced_status_get(p_rxdesc, ENET_RDES4_IPPLDT);
```

### enet\_desc\_select\_enhanced\_mode

The description of enet\_desc\_select\_enhanced\_mode is shown as below:

**Table 3-277. Function enet\_desc\_select\_enhanced\_mode**

<b>Function name</b>	enet_desc_select_enhanced_mode
----------------------	--------------------------------

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

Example:

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

### enet\_ptp\_enhanced\_descriptors\_chain\_init

The description of enet\_ptp\_enhanced\_descriptors\_chain\_init is shown as below:

**Table 3-278. Function enet\_ptp\_enhanced\_descriptors\_chain\_init**

<b>Function name</b>	enet_ptp_enhanced_descriptors_chain_init
<b>Function prototype</b>	void enet_ptp_enhanced_descriptors_chain_init(enet_dmadirection_enum direction);
<b>Function descriptions</b>	initialize the DMA Tx/Rx descriptors's parameters in enhanced chain mode with ptp function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>direction</b>	the descriptors which users want to init, only one parameter can be selected which is shown as below
ENET_DMA_TX	DMA Tx descriptors
ENET_DMA_RX	DMA Rx descriptors
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

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

### enet\_ptp\_enhanced\_descriptors\_ring\_init

The description of enet\_ptp\_enhanced\_descriptors\_ring\_init is shown as below:

**Table 3-279. Function enet\_ptp\_enhanced\_descriptors\_ring\_init**

<b>Function name</b>	enet_ptp_enhanced_descriptors_ring_init
<b>Function prototype</b>	void enet_ptp_enhanced_descriptors_ring_init(enet_dmadirection_enum direction);
<b>Function descriptions</b>	initialize the DMA Tx/Rx descriptors's parameters in enhanced ring mode with ptp function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>direction</b>	the descriptors which users want to init, only one parameter can be selected which is shown as below
ENET_DMA_TX	DMA Tx descriptors
ENET_DMA_RX	DMA Rx descriptors
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## enet\_ptpframe\_receive\_enhanced\_mode

The description of enet\_ptpframe\_receive\_enhanced\_mode is shown as below:

**Table 3-280. Function enet\_ptpframe\_receive\_enhanced\_mode**

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

Example:

```
/* receive a packet data with timestamp values to application buffer in DMA enhanced mode
*/
```

```
uint32_t rx_buffer[500];
```

```
uint32_t time_stamp[2];
```

```
ErrStatus status;
```

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

## enet\_ptpframe\_transmit\_enhanced\_mode

The description of enet\_ptpframe\_transmit\_enhanced\_mode is shown as below:

**Table 3-281. Function enet\_ptpframe\_transmit\_enhanced\_mode**

<b>Function name</b>	enet_ptpframe_transmit_enhanced_mode
<b>Function prototype</b>	ErrStatus enet_ptpframe_transmit_enhanced_mode(uint8_t *buffer, uint32_t

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

Example:

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

```
uint32_t tx_buffer[500];
```

```
uint32_t time_stamp[2];
```

```
ErrStatus status;
```

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

### enet\_desc\_select\_normal\_mode

The description of enet\_desc\_select\_normal\_mode is shown as below:

**Table 3-282. Function enet\_desc\_select\_normal\_mode**

<b>Function name</b>	enet_desc_select_normal_mode
<b>Function prototype</b>	void enet_desc_select_normal_mode(void);
<b>Function descriptions</b>	configure descriptor to work in normal mode
<b>Precondition</b>	-
<b>The called functions</b>	-

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

Example:

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

```
enet_desc_select_normal_mode( );
```

### enet\_ptp\_normal\_descriptors\_chain\_init

The description of enet\_ptp\_normal\_descriptors\_chain\_init is shown as below:

**Table 3-283. Function enet\_ptp\_normal\_descriptors\_chain\_init**

Function name	enet_ptp_normal_descriptors_chain_init
Function prototype	void enet_ptp_normal_descriptors_chain_init(enet_dmadirection_enum direction, enet_descriptors_struct *desc_ptptab);
Function descriptions	initialize the DMA Tx/Rx descriptors's parameters in normal chain mode with PTP function
Precondition	-
The called functions	-
Input parameter{in}	
direction	the descriptors which users want to init only one parameter can be selected which is shown as below
ENET_DMA_TX	DMA Tx descriptors
ENET_DMA_RX	DMA Rx descriptors
Input parameter{in}	
desc_ptptab	pointer to the first descriptor address of PTP Rx descriptor table, the structure members can refer to <a href="#">Structure enet descriptors struct</a>
Output parameter{out}	
-	-
Return value	



-	-
---	---

Example:

```
/* initialize the DMA Rx descriptors's parameters in normal chain mode with PTP function */
enet_descriptors_struct ptp_rxdesc_tab[ENET_RXBUF_NUM];

enet_ptp_normal_descriptors_chain_init(ENET_DMA_RX, ptp_rxdesc_tab);
```

### enet\_ptp\_normal\_descriptors\_ring\_init

The description of enet\_ptp\_normal\_descriptors\_ring\_init is shown as below:

**Table 3-284. Function enet\_ptp\_normal\_descriptors\_ring\_init**

<b>Function name</b>	enet_ptp_normal_descriptors_ring_init
<b>Function prototype</b>	void enet_ptp_normal_descriptors_ring_init(enet_dmadirection_enum direction, enet_descriptors_struct *desc_ptptab);
<b>Function descriptions</b>	initialize the DMA Tx/Rx descriptors's parameters in normal ring mode with PTP function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>direction</b>	the descriptors which users want to init only one parameter can be selected which is shown as below
ENET_DMA_TX	DMA Tx descriptors
ENET_DMA_RX	DMA Rx descriptors
<b>Input parameter{in}</b>	
<b>desc_ptptab</b>	pointer to the first descriptor address of PTP Rx descriptor table, the structure members can refer to <a href="#">Structure enet_descriptors_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the DMA Rx descriptors's parameters in normal ring mode with PTP function */
enet_descriptors_struct ptp_rxdesc_tab[ENET_RXBUF_NUM];
```

```
enet_ptp_normal_descriptors_ring_init(ENET_DMA_RX, ptp_rxdesc_tab);
```

### enet\_ptpframe\_receive\_normal\_mode

The description of enet\_ptpframe\_receive\_normal\_mode is shown as below:

**Table 3-285. Function enet\_ptpframe\_receive\_normal\_mode**

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

Example:

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

### enet\_ptpframe\_transmit\_normal\_mode

The description of enet\_ptpframe\_transmit\_normal\_mode is shown as below:

**Table 3-286. Function enet\_ptpframe\_transmit\_normal\_mode**

<b>Function name</b>	enet_ptpframe_transmit_normal_mode
----------------------	------------------------------------

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

Example:

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

```
uint32_t tx_buffer[500];
```

```
uint32_t time_stamp[2];
```

```
ErrStatus status;
```

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

### enet\_wum\_filter\_register\_pointer\_reset

The description of enet\_wum\_filter\_register\_pointer\_reset is shown as below:

**Table 3-287. Function enet\_wum\_filter\_register\_pointer\_reset**

<b>Function name</b>	enet_wum_filter_register_pointer_reset
<b>Function prototype</b>	void enet_wum_filter_register_pointer_reset(void);
<b>Function descriptions</b>	wakeup frame filter register pointer reset
<b>Precondition</b>	-

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

Example:

```
/* reset wakeup frame filter register pointer */
```

```
enet_wum_filter_register_pointer_reset ();
```

### enet\_wum\_filter\_config

The description of enet\_wum\_filter\_config is shown as below:

**Table 3-288. Function enet\_wum\_filter\_config**

Function name	enet_wum_filter_config
Function prototype	void enet_wum_filter_config(uint32_t pdata[]);
Function descriptions	set the remote wakeup frame registers
Precondition	-
The called functions	-
Input parameter{in}	
pdata	pointer to buffer data which is written to remote wakeup frame registers (8 words total)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the remote wakeup frame registers */
```

```
uint32_t wum_data[8];
```

```
enet_wum_filter_config (wum_data);
```

## enet\_wum\_feature\_enable

The description of enet\_wum\_feature\_enable is shown as below:

**Table 3-289. Function enet\_wum\_feature\_enable**

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

Example:

```
/* enable power down mode */
```

```
enet_wum_feature_enable(ENET_WUM_POWER_DOWN);
```

## enet\_wum\_feature\_disable

The description of enet\_wum\_feature\_disable is shown as below:

**Table 3-290. Function enet\_wum\_feature\_disable**

<b>Function name</b>	enet_wum_feature_disable
<b>Function prototype</b>	void enet_wum_feature_disable(uint32_t feature)

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

Example:

```
/* disable power down mode */
```

```
enet_wum_feature_disable(ENET_WUM_POWER_DOWN);
```

### enet\_msc\_counters\_reset

The description of enet\_msc\_counters\_reset is shown as below:

**Table 3-291. Function enet\_msc\_counters\_reset**

<b>Function name</b>	enet_msc_counters_reset
<b>Function prototype</b>	void enet_msc_counters_reset(void);
<b>Function descriptions</b>	reset the MAC statistics counters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* reset the MAC statistics counters */
```

```
enet_msc_counters_reset();
```

### enet\_msc\_feature\_enable

The description of enet\_msc\_feature\_enable is shown as below:

**Table 3-292. Function enet\_msc\_feature\_enable**

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

Example:

```
/* enable counter stop rollover function */
```

```
enet_msc_feature_enable(ENET_MSC_COUNTER_STOP_ROLLOVER);
```

## enet\_msc\_feature\_disable

The description of enet\_msc\_feature\_disable is shown as below:

**Table 3-293. Function enet\_msc\_feature\_disable**

<b>Function name</b>	enet_msc_feature_disable
<b>Function prototype</b>	void enet_msc_feature_disable(uint32_t feature);
<b>Function descriptions</b>	disable the MAC statistics counter features
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>feature</b>	one or more parameters can be selected which are shown as below
ENET_MSC_COUNTER_STOP_ROLLOVER	counter stop rollover
ENET_MSC_COUNTER_RESET_ON_READ	reset on read
ENET_MSC_COUNTER_FREEZE	MSC counter freeze
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable counter stop rollover function */
```

```
enet_msc_feature_disable(ENET_MSC_COUNTER_STOP_ROLLOVER);
```

## enet\_msc\_counters\_preset\_config

The description of enet\_msc\_counters\_preset\_config is shown as below:

**Table 3-294. Function enet\_msc\_counters\_preset\_config**

<b>Function name</b>	enet_msc_counters_preset_config
<b>Function prototype</b>	void enet_msc_counters_preset_config(enet_msc_preset_enum mode);
<b>Function descriptions</b>	configure MAC statistics counters preset mode
<b>Precondition</b>	-



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

Example:

```
/* preset all MSC counters to almost-half */
```

```
enet_msc_counters_preset_config (ENET_MSC_PRESET_HALF);
```

### enet\_msc\_counters\_get

The description of enet\_msc\_counters\_get is shown as below:

**Table 3-295. Function enet\_msc\_counters\_get**

<b>Function name</b>	enet_msc_counters_get
<b>Function prototype</b>	uint32_t enet_msc_counters_get(enet_msc_counter_enum counter);
<b>Function descriptions</b>	get MAC statistics counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>counter</b>	MSC counters which is selected only one parameter can be selected which is shown as below
<i>ENET_MSC_TX_SCCNT</i>	MSC transmitted good frames after a single collision counter

<i>ENET_MSC_TX_MSC CNT</i>	MSC transmitted good frames after more than a single collision counter
<i>ENET_MSC_TX_TGFC NT</i>	MSC transmitted good frames counter
<i>ENET_MSC_RX_RFCE CNT</i>	MSC received frames with CRC error counter
<i>ENET_MSC_RX_RFAE CNT</i>	MSC received frames with alignment error counter
<i>ENET_MSC_RX_RGU FCNT</i>	MSC received good unicast frames counter
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	the MSC counter value

Example:

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

```
uint32_t reval;
```

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

### enet\_ptp\_feature\_enable

The description of enet\_ptp\_feature\_enable is shown as below:

**Table 3-296. Function enet\_ptp\_feature\_enable**

<b>Function name</b>	enet_ptp_feature_enable
<b>Function prototype</b>	void enet_ptp_feature_enable(uint32_t feature);
<b>Function descriptions</b>	enable the PTP features
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>feature</b>	the feature of ENET PTP mode one or more parameters can be selected which are shown as below
<i>ENET_RXTX_TIMEST AMP</i>	timestamp function for transmit and receive frames

<i>ENET_PTP_TIMESTAMP_INTERRUPT</i>	timestamp interrupt trigger
<i>ENET_ALL_RX_TIMESTAMP</i>	all received frames are taken snapshot
<i>ENET_NONTYPE_FRAME_SNAPSHOT</i>	take snapshot when received non type frame
<i>ENET_IPV6_FRAME_SNAPSHOT</i>	take snapshot for IPv6 frame
<i>ENET_IPV4_FRAME_SNAPSHOT</i>	take snapshot for IPv4 frame
<i>ENET_PTP_FRAME_USE_MACADDRESS_FILTER</i>	use MAC address1-3 to filter the PTP frame
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable PTP function for all received frames */
```

```
enet_ptp_feature_enable(ENET_ALL_RX_TIMESTAMP);
```

### enet\_ptp\_feature\_disable

The description of enet\_ptp\_feature\_disable is shown as below:

**Table 3-297. Function enet\_ptp\_feature\_disable**

<b>Function name</b>	enet_ptp_feature_disable
<b>Function prototype</b>	void enet_ptp_feature_disable(uint32_t feature);
<b>Function descriptions</b>	disable the PTP features
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>feature</b>	the feature of ENET PTP mode one or more parameters can be selected which are shown as below

<i>ENET_RXTX_TIMESTAMP</i>	timestamp function for transmit and receive frames
<i>ENET_PTP_TIMESTAMP_INTERRUPT</i>	timestamp interrupt trigger
<i>ENET_ALL_RX_TIMESTAMP</i>	all received frames are taken snapshot
<i>ENET_NONTYPE_FRAME_SNAPSHOT</i>	take snapshot when received non type frame
<i>ENET_IPV6_FRAME_SNAPSHOT</i>	take snapshot for IPv6 frame
<i>ENET_IPV4_FRAME_SNAPSHOT</i>	take snapshot for IPv4 frame
<i>ENET_PTP_FRAME_USE_MACADDRESS_FILTER</i>	use MAC address1-3 to filter the PTP frame
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable PTP function for all received frames */
```

```
enet_ptp_feature_disable(ENET_ALL_RX_TIMESTAMP);
```

### enet\_ptp\_timestamp\_function\_config

The description of enet\_ptp\_timestamp\_function\_config is shown as below:

**Table 3-298. Function enet\_ptp\_timestamp\_function\_config**

<b>Function name</b>	enet_ptp_timestamp_function_config
<b>Function prototype</b>	ErrStatus enet_ptp_timestamp_function_config(enet_ptp_function_enum func);
<b>Function descriptions</b>	configure the PTP timestamp function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>func</b>	only one parameter can be selected which is shown as below
<i>ENET_CKNT_ORDINARY</i>	type of ordinary clock node type for timestamp
<i>ENET_CKNT_BOUNDARY</i>	type of boundary clock node type for timestamp
<i>ENET_CKNT_END_TO_END</i>	type of end-to-end transparent clock node type for timestamp
<i>ENET_CKNT_PEER_TO_PEER</i>	type of peer-to-peer transparent clock node type for timestamp
<i>ENET_PTP_ADDEND_UPDATE</i>	addend register update
<i>ENET_PTP_SYSTIME_UPDATE</i>	timestamp update
<i>ENET_PTP_SYSTIME_INIT</i>	timestamp initialize
<i>ENET_PTP_FINEMODE</i>	the system timestamp uses the fine method for updating
<i>ENET_PTP_COARSEMODE</i>	the system timestamp uses the coarse method for updating
<i>ENET_SUBSECOND_DIGITAL_ROLLOVER</i>	digital rollover mode
<i>ENET_SUBSECOND_BINARY_ROLLOVER</i>	digital rollover mode
<i>ENET_SNOOPING_PTP_VERSION_2</i>	version 2
<i>ENET_SNOOPING_PTP_VERSION_1</i>	version 1
<i>ENET_EVENT_TYPE_MESSAGES_SNAPSHOT</i>	only event type messages are taken snapshot
<i>ENET_ALL_TYPE_MESSAGES_SNAPSHOT</i>	all type messages are taken snapshot except announce, management and signaling message
<i>ENET_MASTER_NODE_MESSAGE_SNAPSHOT</i>	snapshot is only take for master node message

<i>ENET_SLAVE_NODE_MESSAGE_SNAPSHOT</i> <i>T</i>	snapshot is only taken for slave node message
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* config addend register update function */
```

```
ErrStatus reval = ERROR;
```

```
reval = enet_ptp_timestamp_function_config(ENET_PTP_ADDEND_UPDATE);
```

### enet\_ptp\_subsecond\_increment\_config

The description of enet\_ptp\_subsecond\_increment\_config is shown as below:

**Table 3-299. Function enet\_ptp\_subsecond\_increment\_config**

<b>Function name</b>	enet_ptp_subsecond_increment_config
<b>Function prototype</b>	void enet_ptp_subsecond_increment_config(uint32_t subsecond);
<b>Function descriptions</b>	configure system time subsecond increment value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>subsecond</b>	the value will be added to the subsecond value of system time, this value must be between 0 and 0xFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure 0x1F as system time subsecond increment value */
```

```
enet_ptp_subsecond_increment_config(0x1F);
```

## enet\_ptp\_timestamp\_addend\_config

The description of enet\_ptp\_timestamp\_addend\_config is shown as below:

**Table 3-300. Function enet\_ptp\_timestamp\_addend\_config**

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

```
/* added 0x1FFF to the accumulator register */
```

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

## enet\_ptp\_timestamp\_update\_config

The description of enet\_ptp\_timestamp\_update\_config is shown as below:

**Table 3-301. Function enet\_ptp\_timestamp\_update\_config**

<b>Function name</b>	enet_ptp_timestamp_update_config
<b>Function prototype</b>	void enet_ptp_timestamp_update_config(uint32_t sign, uint32_t second, uint32_t subsecond);
<b>Function descriptions</b>	initialize or add/subtract to second of the system time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sign</b>	timestamp update positive or negative sign, only one parameter can be selected which is shown as below
<b>ENET_PTP_ADD_TO_</b>	update value is added to system time

<i>TIME</i>	
<i>ENET_PTP_SUBSTRA CT_FROM_TIME</i>	timestamp update value is subtracted from system time
<b>Input parameter{in}</b>	
<b>second</b>	initializing or adding/subtracting to second of the system time (0 – 0xFFFF FFFF)
<b>Input parameter{in}</b>	
<b>subsecond</b>	the current subsecond of the system time with 0.46 ns accuracy (0 – 0x7FFF FFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### enet\_ptp\_expected\_time\_config

The description of enet\_ptp\_expected\_time\_config is shown as below:

**Table 3-302. Function enet\_ptp\_expected\_time\_config**

<b>Function name</b>	enet_ptp_expected_time_config
<b>Function prototype</b>	void enet_ptp_expected_time_config(uint32_t second, uint32_t nanosecond);
<b>Function descriptions</b>	configure the expected target time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>second</b>	the expected target second time (0 – 0xFFFF FFFF)
<b>Input parameter{in}</b>	
<b>nanosecond</b>	the expected target nanosecond time (signed) (0 – 0xFFFF FFFF)
<b>Output parameter{out}</b>	



-	-
Return value	
-	-

Example:

```
/* configure the expected target time */
enet_ptp_expected_time_config(2000, 0);
```

### enet\_ptp\_system\_time\_get

The description of enet\_ptp\_system\_time\_get is shown as below:

**Table 3-303. Function enet\_ptp\_system\_time\_get**

<b>Function name</b>	enet_ptp_system_time_get
<b>Function prototype</b>	void enet_ptp_system_time_get(enet_ptp_systime_struct *systime_struct);
<b>Function descriptions</b>	get the current system time
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
<b>systime_struct</b>	pointer to a enet_ptp_systime_struct structure which contains parameters of PTP system time, the structure members can refer to <a href="#">Structure enet_ptp_systime_struct</a>
Return value	
-	-

Example:

```
/* get the current system time */
enet_ptp_systime_struct systime;
enet_ptp_system_time_get(&systime);
```

### enet\_ptp\_pps\_output\_frequency\_config

The description of enet\_ptp\_pps\_output\_frequency\_config is shown as below:

Table 3-304. Function enet\_ptp\_pps\_output\_frequency\_config

<b>Function name</b>	enet_ptp_pps_output_frequency_config
<b>Function prototype</b>	void enet_ptp_pps_output_frequency_config(uint32_t freq);
<b>Function descriptions</b>	configure the PPS output frequency
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>freq</b>	
ENET_PPISOFC_1HZ	PPS output 1Hz frequency
ENET_PPISOFC_2HZ	PPS output 2Hz frequency
ENET_PPISOFC_4HZ	PPS output 4Hz frequency
ENET_PPISOFC_8HZ	PPS output 8Hz frequency
ENET_PPISOFC_16HZ	PPS output 16Hz frequency
ENET_PPISOFC_32HZ	PPS output 32Hz frequency
ENET_PPISOFC_64HZ	PPS output 64Hz frequency
ENET_PPISOFC_128HZ	PPS output 128Hz frequency
ENET_PPISOFC_256HZ	PPS output 256Hz frequency
ENET_PPISOFC_512HZ	PPS output 512Hz frequency
ENET_PPISOFC_1024HZ	PPS output 1024Hz frequency
ENET_PPISOFC_2048HZ	PPS output 2048Hz frequency
ENET_PPISOFC_4096HZ	PPS output 4096Hz frequency
ENET_PPISOFC_8192HZ	PPS output 8192Hz frequency
ENET_PPISOFC_16384HZ	PPS output 16384Hz frequency
ENET_PPISOFC_32768	PPS output 32768Hz frequency

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

Example:

```
/* configure the PPS output frequency as 1Hz */
```

```
enet_ptp_pps_output_frequency_config(ENET_PPISOFC_1HZ);
```

### enet\_initpara\_reset

The description of enet\_initpara\_reset is shown as below:

**Table 3-305. Function enet\_initpara\_reset**

<b>Function name</b>	enet_initpara_reset
<b>Function prototype</b>	void enet_initpara_reset(void);
<b>Function descriptions</b>	reset the ENET initpara struct, call it before using enet_initpara_config()
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the ENET initpara struct */
```

```
enet_initpara_reset();
```

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

### 3.11.1. Descriptions of Peripheral registers

EXMC registers are listed in the table shown as below:

**Table 3-306. EXMC Registers**

Registers	Descriptions
EXMC_SNCTL	SRAM/NOR Flash control registers
EXMC_SNTCFG	SRAM/NOR Flash timing configuration registers
EXMC_SNWTCFG	SRAM/NOR Flash write timing configuration registers
EXMC_NPCTL	NAND flash/PC card control registers
EXMC_NPINTEN	NAND flash/PC card interrupt enable registers
EXMC_NPCTCFG	NAND flash/PC card common space timing configuration registers
EXMC_NPATCFG	NAND flash/PC card attribute space timing configuration registers
EXMC_PIOTCFG3	PC card I/O space timing configuration register
EXMC_NECC	NAND flash ECC registers
EXMC_SDCTL	SDRAM control registers
EXMC_SDTCFG	SDRAM timing configuration registers
EXMC_SDCMD	SDRAM command register
EXMC_SDARI	SDRAM auto-refresh interval register
EXMC_SDSTAT	SDRAM status register
EXMC_SDRSCTL	SDRAM read sample control register
EXMC_SINIT	SPI initialization register
EXMC_SRCMD	SPI read command register
EXMC_SWCMD	SPI write command register
EXMC_SIDL	SPI ID low register
EXMC_SIDH	SPI ID high register

### 3.11.2. Descriptions of Peripheral functions

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

**Table 3-307. 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/PSRAM 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

Function name	Function description
exmc_nand_disable	disable EXMC NAND bankx
exmc_pccard_deinit	deinitialize EXMC PC card bank
exmc_pccard_struct_para_init	initialize exmc_pccard_parameter_struct with the default values
exmc_pccard_init	initialize EXMC PC card bank
exmc_pccard_enable	enable EXMC PC card bank
exmc_pccard_disable	disable EXMC PC card bank
exmc_sdram_deinit	deinitialize EXMC SDRAM device
exmc_sdram_struct_para_init	initialize exmc_sdram_parameter_struct with the default values
exmc_sdram_init	initialize EXMC SDRAM device
exmc_sdram_struct_command_parameter_init	initialize exmc_sdram_command_parameter_struct with the default values
exmc_sqpsram_deinit	deinitialize EXMC SQPSRAM
exmc_sqpsram_struct_para_init	initialize exmc_sqpsram_parameter_struct with the default values
exmc_sqpsram_init	initialize EXMC SQPSRAM
exmc_norsram_consecutive_clock_config	configure consecutive clock
exmc_norsram_page_size_config	configure CRAM page size
exmc_nand_ecc_config	enable or disable the EXMC NAND ECC function
exmc_ecc_get	get the EXMC ECC value
exmc_sdram_readsample_enable	enable or disable read sample
exmc_sdram_readsample_config	configure the delayed sample clock of read data
exmc_sdram_command_config	configure the SDRAM memory command
exmc_sdram_refresh_count_set	set auto-refresh interval
exmc_sdram_autorefresh_number_set	set the number of successive auto-refresh command
exmc_sdram_write_protection_config	config the write protection function
exmc_sdram_bankstatus_get	get the status of SDRAM device0 or device1
exmc_sqpsram_read_command_set	set the read command
exmc_sqpsram_write_command_set	set the write command
exmc_sqpsram_read_id_command_send	send SPI read ID command
exmc_sqpsram_write_cmd_send	send SPI special command which does not have address and data phase
exmc_sqpsram_low_id_get	get the EXMC SPI ID low data
exmc_sqpsram_high_id_get	get the EXMC SPI ID high data
exmc_sqpsram_send_command_status_get	get the bit value of EXMC send write command bit or read ID command
exmc_interrupt_enable	enable EXMC interrupt

Function name	Function description
exmc_interrupt_disable	disable EXMC interrupt
exmc_flag_get	get EXMC flag status
exmc_flag_clear	clear EXMC flag status
exmc_interrupt_flag_get	get EXMC interrupt flag
exmc_interrupt_flag_clear	clear EXMC interrupt flag

### Structure exmc\_norsram\_timing\_parameter\_struct

**Table 3-308. Structure exmc\_norsram\_timing\_parameter\_struct**

Member name	Function description
asyn_access_mode	asynchronous access mode
syn_data_latency	configure the data latency
syn_clk_division	configure the clock divide ratio
bus_latency	configure the bus latency
asyn_data_setup_time	configure the data setup time, asynchronous access mode valid
asyn_address_hold_time	configure the address hold time, asynchronous access mode valid
asyn_address_setup_time	configure the data setup time, asynchronous access mode valid

### Structure exmc\_norsram\_parameter\_struct

**Table 3-309. Structure exmc\_norsram\_parameter\_struct**

Member name	Function description
norsram_region	select the region of EXMC NOR/SRAM
write_mode	the write mode, synchronous mode or asynchronous mode
extended_mode	enable or disable the extended mode
asyn_wait	enable or disable the asynchronous wait function
nwait_signal	enable or disable the NWAIT signal while in synchronous burst mode
memory_write	enable or disable the write operation
nwait_config	NWAIT signal configuration
wrap_burst_mode	enable or disable the wrap burst mode
nwait_polarity	specifies the polarity of NWAIT signal from memory
burst_mode	enable or disable the burst mode
databus_width	specifies the databus width of external memory
memory_type	specifies the type of external memory
address_data_mux	specifies whether the data bus and address bus are multiplexed
read_write_timing	timing parameters for read and write if the extended mode is not used or the timing parameters for read if the extended mode is used
write_timing	timing parameters for write when the extended mode is used

### Structure `exmc_nand_pccard_timing_parameter_struct`

**Table 3-310. Structure `exmc_nand_pccard_timing_parameter_struct`**

Member name	Function description
<code>databus_hiztime</code>	configure the databus HiZ time for write operation
<code>holdtime</code>	configure the address hold time(or the data hold time for write operation)
<code>waittime</code>	configure the minimum wait time
<code>setuptime</code>	configure the address setup time

### Structure `exmc_nand_parameter_struct`

**Table 3-311. Structure `exmc_nand_parameter_struct`**

Member name	Function description
<code>nand_bank</code>	select the bank of NAND
<code>ecc_size</code>	the page size for the ECC calculation
<code>atr_latency</code>	configure the latency of ALE low to RB low
<code>ctr_latency</code>	configure the latency of CLE low to RB low
<code>ecc_logic</code>	enable or disable the ECC calculation logic
<code>databus_width</code>	the NAND flash databus width
<code>wait_feature</code>	enable or disable the wait feature
<code>common_space_timing</code>	the timing parameters for NAND flash common space
<code>attribute_space_timing</code>	the timing parameters for NAND flash attribute space

### Structure `exmc_pccard_parameter_struct`

**Table 3-312. Structure `exmc_pccard_parameter_struct`**

Member name	Function description
<code>atr_latency</code>	configure the latency of ALE low to RB low
<code>ctr_latency</code>	configure the latency of CLE low to RB low
<code>wait_feature</code>	enable or disable the wait feature
<code>common_space_timing</code>	the timing parameters for PC card common space
<code>attribute_space_timing</code>	the timing parameters for PC card attribute space
<code>io_space_timing</code>	the timing parameters for PC card IO space

### Structure `exmc_sdram_timing_parameter_struct`

**Table 3-313. Structure `exmc_sdram_timing_parameter_struct`**

Member name	Function description
<code>row_to_column_delay</code>	configure the row to column delay
<code>row_precharge_delay</code>	configure the row precharge delay

Member name	Function description
y	
write_recovery_delay	configure the write recovery delay
auto_refresh_delay	configure the auto refresh delay
row_address_select_delay	configure the row address select delay
exit_selfrefresh_delay	configure the exit self-refresh delay
load_mode_register_delay	configure the load mode register delay

### Structure exmc\_sdram\_parameter\_struct

**Table 3-314. Structure exmc\_sdram\_parameter\_struct**

Member name	Function description
sdram_device	select the device of SDRAM
pipeline_read_delay	the delay for reading data after CAS latency in HCLK clock cycles
burst_read_switch	enable or disable the burst read
sdclk_config	the SDCLK memory clock for both SDRAM banks
write_protection	enable or disable SDRAM bank write protection function
cas_latency	configure the SDRAM CAS latency
internal_bank_number	the number of internal bank
data_width	the databus width of SDRAM memory
row_address_width	the bit width of a row address
column_address_width	the bit width of a column address
timing	the timing parameters for write and read SDRAM

### Structure exmc\_sdram\_command\_parameter\_struct

**Table 3-315. Structure exmc\_sdram\_command\_parameter\_struct**

Member name	Function description
mode_register_content	the SDRAM mode register content
auto_refresh_number	the number of successive auto-refresh cycles will be send when CMD = 011
bank_select	the bank which command will be sent to
command	the commands that will be sent to SDRAM



## Structure exmc\_sqpsram\_parameter\_struct

**Table 3-316. Structure exmc\_sqpsram\_parameter\_struct**

Member name	Function description
sample_polarity	read data sample polarity
id_length	SPI PSRAM ID length
address_bits	bit number of SPI PSRAM address phase
command_bits	bit number of SPI PSRAM command phase

## exmc\_norsram\_deinit

The description of exmc\_norsram\_deinit is shown as below:

**Table 3-317. Function exmc\_norsram\_deinit**

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

Example:

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

```
exmc_norsram_deinit(EXMC_BANK0_NORSRAM_REGION1);
```

## exmc\_norsram\_struct\_para\_init

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

**Table 3-318. 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 3-309</a> . <a href="#">Structure exmc_norsram_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-319. Function exmc\_norsram\_init**

<b>Function name</b>	exmc_norsram_init
<b>Function prototype</b>	void exmc_norsram_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
<b>Function descriptions</b>	initialize EXMC NOR/SRAM region
<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 3-309</a> . <a href="#">Structure exmc_norsram_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize EXMC NOR/SRAM bank */

exmc_norsram_parameter_struct lcd_init_struct;

exmc_norsram_timing_parameter_struct lcd_timing_init_struct;

/* configure timing parameter */

lcd_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;

lcd_timing_init_struct.syn_data_latency = EXMC_DATA_LAT_2_CLK;

lcd_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;

lcd_timing_init_struct.bus_latency = 1;
```

```

lcd_timing_init_struct.asyn_data_setuptime = 5;

lcd_timing_init_struct.asyn_address_holdtime = 2;

lcd_timing_init_struct.asyn_address_setuptime = 2;

/* configure EXMC bus parameters */

lcd_init_struct.norsram_region = EXMC_BANK0_NORSRAM_REGION1;

lcd_init_struct.write_mode = EXMC_ASYN_WRITE;

lcd_init_struct.extended_mode = DISABLE;

lcd_init_struct.asyn_wait = DISABLE;

lcd_init_struct.nwait_signal = DISABLE;

lcd_init_struct.memory_write = ENABLE;

lcd_init_struct.nwait_config = EXMC_NWAIT_CONFIG_BEFORE;

lcd_init_struct.wrap_burst_mode = DISABLE;

lcd_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;

lcd_init_struct.burst_mode = DISABLE;

lcd_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;

lcd_init_struct.memory_type = EXMC_MEMORY_TYPE_SRAM;

lcd_init_struct.address_data_mux = DISABLE;

lcd_init_struct.read_write_timing = &lcd_timing_init_struct;

lcd_init_struct.write_timing = &lcd_timing_init_struct;

exmc_norsram_init(&lcd_init_struct);

```

### exmc\_norsram\_enable

The description of exmc\_norsram\_enable is shown as below:

**Table 3-320. Function exmc\_norsram\_enable**

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

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

Example:

```
/* enable the EXMC NOR/SRAM region1 of bank0 */
exmc_norsram_enable(EXMC_BANK0_NORSRAM_REGION1);
```

### exmc\_norsram\_disable

The description of exmc\_norsram\_disable is shown as below:

**Table 3-321. Function exmc\_norsram\_disable**

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

Example:

```
/* disable the EXMC NOR/SRAM region1 of bank0 */
exmc_norsram_disable(EXMC_BANK0_NORSRAM_REGION1);
```

### exmc\_nand\_deinit

The description of exmc\_nand\_deinit is shown as below:

**Table 3-322. Function exmc\_nand\_deinit**

<b>Function name</b>	exmc_nand_deinit
<b>Function prototype</b>	void exmc_nand_deinit(uint32_t exmc_nand_bank);
<b>Function descriptions</b>	deinitialize EXMC NAND bank
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>exmc_nand_bank</b>	the bank of NAND
<i>EXMC_BANKx_NAND</i>	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
exmc_norsram_deinit(EXMC_BANK1_NAND);
```

### exmc\_nand\_struct\_para\_init

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

**Table 3-323. 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}	
<b>exmc_nand_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-311. 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 3-324. Function exmc\_nand\_init**

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

<b>Function descriptions</b>	initialize EXMC NAND bank
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_nand_init_struct</b> <b>t</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-311</a> . <a href="#">Structure exmc_nand_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-325. Function exmc\_nand\_enable**

<b>Function name</b>	exmc_nand_enable
<b>Function prototype</b>	void exmc_nand_enable(uint32_t exmc_nand_bank);

<b>Function descriptions</b>	enable EXMC NAND bank
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_nand_bank</b>	the bank of NAND
<i>EXMC_BANKx_NAND</i>	x=1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable EXMC NAND bank1 */
exmc_nand_enable(EXMC_BANK1_NAND);
```

### exmc\_nand\_disable

The description of exmc\_nand\_disable is shown as below:

**Table 3-326. Function exmc\_nand\_disable**

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

Example:

```
/* disable EXMC NAND bank1 */
exmc_nand_disable(EXMC_BANK1_NAND);
```

### exmc\_pccard\_deinit

The description of exmc\_pccard\_deinit is shown as below:

**Table 3-327. Function exmc\_pccard\_deinit**

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

<b>Function descriptions</b>	deinitialize EXMC PC card bank
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* deinitialize EXMC PC card bank */
exmc_pccard_deinit();
```

### exmc\_pccard\_struct\_para\_init

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

**Table 3-328. Function exmc\_pccard\_struct\_para\_init**

<b>Function name</b>	exmc_pccard_struct_para_init
<b>Function prototype</b>	void exmc_pccard_struct_para_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);
<b>Function descriptions</b>	initialize the struct exmc_pccard_parameter_struct
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_pccard_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">_</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the struct pccard_init_struct */
exmc_pccard_parameter_struct pccard_init_struct;
exmc_pccard_struct_para_init (&pccard_init_struct);
```

### exmc\_pccard\_init

The description of exmc\_pccard\_init is shown as below:

**Table 3-329. Function exmc\_pccard\_init**

<b>Function name</b>	exmc_pccard_init
----------------------	------------------



<b>Function prototype</b>	void exmc_pccard_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);
<b>Function descriptions</b>	initialize EXMC PC card bank
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_pccard_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-312. Structure exmc_pccard_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
exmc_pccard_parameter_struct pccard_init_struct;

exmc_nand_pccard_timing_parameter_struct pccard_timing_init_struct;

/* EXMC configuration */

pccard_timing_init_struct.setuptime = 5;

pccard_timing_init_struct.waittime = 4;

pccard_timing_init_struct.holdtime = 2;

pccard_timing_init_struct.databus_hiztime = 2;

pccard_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;

pccard_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;

pccard_init_struct.wait_feature = ENABLE;

pccard_init_struct.common_space_timing = & pccard_timing_init_struct;

pccard_init_struct.attribute_space_timing = & pccard_timing_init_struct;

pccard_init_struct.io_space_timing = & pccard_timing_init_struct;

exmc_pccard_init(&pccard_init_struct);
```

### exmc\_pccard\_enable

The description of exmc\_pccard\_enable is shown as below:

**Table 3-330. Function exmc\_pccard\_enable**

<b>Function name</b>	exmc_pccard_enable
<b>Function prototype</b>	void exmc_pccard_enable(void);
<b>Function descriptions</b>	enable EXMC PC card bank
<b>Precondition</b>	-

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

Example:

```
/* enable EXMC PC card bank */
```

```
exmc_pccard_enable();
```

### exmc\_pccard\_disable

The description of exmc\_pccard\_disable is shown as below:

**Table 3-331. Function exmc\_pccard\_disable**

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

Example:

```
/* disable EXMC PC card bank */
```

```
exmc_pccard_disable();
```

### exmc\_sdram\_deinit

The description of exmc\_sdram\_deinit is shown as below:

**Table 3-332. Function exmc\_sdram\_deinit**

Function name	exmc_sdram_deinit
Function prototype	void exmc_sdram_deinit(uint32_t exmc_sdram_device);
Function descriptions	deinitialize EXMC SDRAM device
Precondition	-
The called functions	-
Input parameter{in}	

<b>exmc_sdram_device</b>	EXMC SDRAM device
<i>EXMC_SDRAM_DEVICE</i> <i>Ex</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* deinitialize EXMC SDRAM device1 */
exmc_sdram_deinit(EXMC_SDRAM_DEVICE1);
```

### exmc\_sdram\_struct\_para\_init

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

**Table 3-333. Function exmc\_sdram\_struct\_para\_init**

<b>Function name</b>	exmc_sdram_struct_para_init
<b>Function prototype</b>	exmc_sdram_struct_para_init(exmc_sdram_parameter_struct* exmc_sdram_init_struct);
<b>Function descriptions</b>	initialize the struct exmc_sdram_parameter_struct
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_sdram_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-314. Structure exmc_sdram_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the struct sdram_init_struct */
exmc_sdram_parameter_struct sdram_init_struct;
exmc_sdram_struct_para_init (&sdram_init_struct);
```

### exmc\_sdram\_init

The description of exmc\_sdram\_init is shown as below:

**Table 3-334. Function exmc\_sdram\_init**

<b>Function name</b>	exmc_sdram_init
<b>Function prototype</b>	void exmc_sdram_init(exmc_sdram_parameter_struct* exmc_sdram_init_struct);

<b>Function descriptions</b>	initialize EXMC SDRAM device
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_sdram_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-314</a> . <a href="#">Structure exmc_sdram_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

exmc_sdram_parameter_struct      sdram_init_struct;

exmc_sdram_timing_parameter_struct  sdram_timing_init_struct;

/* EXMC configuration */

sdram_timing_init_struct.load_mode_register_delay = 2;

/* XSRD: min = 67ns */

sdram_timing_init_struct.exit_selfrefresh_delay = 7;

/* RASD: min=42ns , max=120k (ns) */

sdram_timing_init_struct.row_address_select_delay = 5;

/* ARFD: min=60ns */

sdram_timing_init_struct.auto_refresh_delay = 6;

/* WRD:  min=1 Clock cycles +6ns */

sdram_timing_init_struct.write_recovery_delay = 2;

/* RPD:  min=18ns */

sdram_timing_init_struct.row_precharge_delay = 2;

/* RCD:  min=18ns */

sdram_timing_init_struct.row_to_column_delay = 2;

sdram_init_struct.sdram_device = sdram_device;

sdram_init_struct.column_address_width = EXMC_SDRAM_COW_ADDRESS_9;

sdram_init_struct.row_address_width = EXMC_SDRAM_ROW_ADDRESS_13;

sdram_init_struct.data_width = EXMC_SDRAM_DATABUS_WIDTH_16B;

sdram_init_struct.internal_bank_number = EXMC_SDRAM_4_INTER_BANK;

sdram_init_struct.cas_latency = EXMC_CAS_LATENCY_3_SDCLK;

```

```

sdram_init_struct.write_protection = DISABLE;

sdram_init_struct.sdclk_config = EXMC_SDCLK_PERIODS_2_HCLK;

sdram_init_struct.burst_read_switch = ENABLE;

sdram_init_struct.pipeline_read_delay = EXMC_PIPELINE_DELAY_1_HCLK;

sdram_init_struct.timing = &sdram_timing_init_struct;

/* EXMC SDRAM bank initialization */

exmc_sdram_init(&sdram_init_struct);

```

### exmc\_sdram\_struct\_command\_para\_init

The description of exmc\_sdram\_struct\_command\_para\_init is shown as below:

**Table 3-335. Function exmc\_sdram\_struct\_command\_para\_init**

Function name	exmc_sdram_struct_command_para_init
Function prototype	void exmc_sdram_struct_command_para_init(exmc_sdram_command_parameter_struct *exmc_sdram_command_init_struct);
Function descriptions	initialize exmc_sdram_command_parameter_struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
exmc_sdram_command_parameter_struct	structure for initialization, the structure members can refer to <a href="#">Table 3-315. Structure exmc_sdram_command_parameter_struct</a>
Return value	
-	-

Example:

```

/* initialize the structure exmc_sdram_command_init_struct */

exmc_sdram_command_parameter_struct t_exmc_sdram_command_init_struct;

exmc_sdram_struct_command_para_init (&exmc_sdram_command_init_struct);

```

### exmc\_sqpsram\_deinit

The description of exmc\_sqpsram\_deinit is shown as below:

**Table 3-336. Function exmc\_sqpsram\_deinit**

Function name	exmc_sqpsram_deinit
Function prototype	void exmc_sqpsram_deinit(void);
Function descriptions	deinitialize EXMC SQPIPSRAM

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

Example:

```
/* deinitialize EXMC SQPIPSRAM */
```

```
exmc_sqipsram_deinit(void);
```

### exmc\_sqipsram\_struct\_para\_init

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

**Table 3-337. Function exmc\_sqipsram\_struct\_para\_init**

<b>Function name</b>	exmc_sqipsram_struct_para_init
<b>Function prototype</b>	void exmc_sqipsram_struct_para_init(exmc_sqipsram_parameter_struct* exmc_sqipsram_init_struct);
<b>Function descriptions</b>	initialize the struct exmc_sqipsram_parameter_struct
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_sqipsram_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-316. Structure exmc_sqipsram_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the struct sqipsram_init_struct */
```

```
exmc_sqipsram_parameter_struct sqipsram_init_struct;
```

```
exmc_sqipsram_struct_para_init (&sqipsram_init_struct);
```

### exmc\_sqipsram\_init

The description of exmc\_sqipsram\_init is shown as below:

**Table 3-338. Function exmc\_sqipsram\_init**

<b>Function name</b>	exmc_sqipsram_init
<b>Function prototype</b>	void exmc_sqipsram_init(exmc_sqipsram_parameter_struct*

	exmc_sqpsram_init_struct);
<b>Function descriptions</b>	initialize EXMC SQPIPSRAM
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_sqpsram_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-316. Structure exmc_sqpsram_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
exmc_sqpsram_parameter_struct      sqpsram_init_struct;

/* EXMC configuration */

sqpsram_init_struct.sample_polarity = EXMC_SQPIPSRAM_SAMPLE_RISING_EDGE;
sqpsram_init_struct.id_length = EXMC_SQPIPSRAM_ID_LENGTH_64B;
sqpsram_init_struct.address_bits = EXMC_SQPIPSRAM_ADDR_LENGTH_24B;
sqpsram_init_struct.command_bits = EXMC_SQPIPSRAM_COMMAND_LENGTH_8B;

/* EXMC SDRAM bank initialization */

exmc_sqpsram_init(&sqpsram_init_struct);
```

### exmc\_norsram\_consecutive\_clock\_config

The description of exmc\_norsram\_consecutive\_clock\_config is shown as below:

**Table 3-339. Function exmc\_norsram\_consecutive\_clock\_config**

<b>Function name</b>	exmc_norsram_consecutive_clock_config
<b>Function prototype</b>	void exmc_norsram_consecutive_clock_config(uint32_t clock_mode);
<b>Function descriptions</b>	configure consecutive clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>clock_mode</b>	the mode of consecutive clock
<i>EXMC_CLOCK_SYN_MODE</i>	the clock is generated only during synchronous access
<i>EXMC_CLOCK_UNCONDITIONALLY</i>	the clock is generated unconditionally
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* configure consecutive clock */
```

```
exmc_norsram_consecutive_clock_config(EXMC_CLOCK_SYN_MODE);
```

### exmc\_norsram\_page\_size\_config

The description of exmc\_norsram\_page\_size\_config is shown as below:

**Table 3-340. 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
EXMC_CRAM_AUTO_SPLIT	the clock is generated only during synchronous access
EXMC_CRAM_PAGE_SIZE_128_BYTES	page size is 128 bytes
EXMC_CRAM_PAGE_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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure CRAM page size */
```

```
exmc_norsram_page_size_config (EXMC_CRAM_PAGE_SIZE_128_BYTES);
```

### exmc\_nand\_ecc\_config

The description of exmc\_nand\_ecc\_config is shown as below:

**Table 3-341. Function exmc\_nand\_ecc\_config**

<b>Function name</b>	exmc_nand_ecc_config
<b>Function prototype</b>	void exmc_nand_ecc_config(uint32_t exmc_nand_bank, ControlStatus



	newvalue);
<b>Function descriptions</b>	enable or disable the EXMC NAND ECC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_nand_bank</b>	specifie the NAND bank
<b>EXMC_BANKx_NAND</b>	x=1,2
<b>Input parameter{in}</b>	
<b>newvalue</b>	ENABLE or DISABLE
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
exmc_nand_ecc_config(EXMC_BANK1_NAND, ENABLE);
```

### exmc\_ecc\_get

The description of exmc\_ecc\_get is shown as below:

**Table 3-342. 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>	
<b>exmc_nand_bank</b>	specifie the NAND bank
<b>EXMC_BANKx_NAND</b>	x=1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	the error correction code(ECC) value

Example:

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

```
uint32_t ecc_value;
```

```
ecc_value = exmc_ecc_get(EXMC_BANK1_NAND);
```

## exmc\_sdram\_readsample\_enable

The description of exmc\_sdram\_readsample\_enable is shown as below:

**Table 3-343. Function exmc\_sdram\_readsample\_enable**

<b>Function name</b>	exmc_sdram_readsample_enable
<b>Function prototype</b>	void exmc_sdram_readsample_enable(ControlStatus newvalue);
<b>Function descriptions</b>	enable or disable read sample
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>newvalue</b>	ENABLE or DISABLE
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable read sample */
```

```
exmc_sdram_readsample_enable(ENABLE);
```

## exmc\_sdram\_readsample\_config

The description of exmc\_sdram\_readsample\_config is shown as below:

**Table 3-344. Function exmc\_sdram\_readsample\_config**

<b>Function name</b>	exmc_sdram_readsample_config
<b>Function prototype</b>	void exmc_sdram_readsample_config(uint32_t delay_cell, uint32_t extra_hclk);
<b>Function descriptions</b>	configure the delayed sample clock of read data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>delay_cell</b>	SDRAM the delayed sample clock of read data
EXMC_SDRAM_X_DELAY_CELL	x=0...15
<b>Input parameter{in}</b>	
<b>extra_hclk</b>	sample cycle of read data
EXMC_SDRAM_READ_SAMPLE_X_EXTRAHCLK	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* configure the delayed sample clock and sample cycle of read data */
```

```
exmc_sdram_readsample_config(EXMC_SDRAM_1_DELAY_CELL,
EXMC_SDRAM_READSAMPLE_1_EXTRAHCLK);
```

### exmc\_sdram\_command\_config

The description of exmc\_sdram\_command\_config is shown as below:

**Table 3-345. Function exmc\_sdram\_command\_config**

<b>Function name</b>	exmc_sdram_command_config
<b>Function prototype</b>	void exmc_sdram_command_config(exmc_sdram_command_parameter_struct* exmc_sdram_command_init_struct);
<b>Function descriptions</b>	configure the SDRAM memory command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_sdram_command_init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-315. Structure exmc_sdram_command_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the SDRAM memory command */
```

```
exmc_sdram_command_parameter_struct    sdram_command_init_struct;
```

```
sdram_command_init_struct.command = EXMC_SDRAM_CLOCK_ENABLE;
```

```
sdram_command_init_struct.bank_select = bank_select;
```

```
sdram_command_init_struct.auto_refresh_number =  
EXMC_SDRAM_AUTO_REFRESH_1_SDCLK;
```

```
sdram_command_init_struct.mode_register_content = 0;
```

```
exmc_sdram_command_config(&sdram_command_init_struct);
```

### exmc\_sdram\_refresh\_count\_set

The description of exmc\_sdram\_refresh\_count\_set is shown as below:

**Table 3-346. Function exmc\_sdram\_refresh\_count\_set**

<b>Function name</b>	exmc_sdram_refresh_count_set
<b>Function prototype</b>	void exmc_sdram_refresh_count_set(uint32_t exmc_count);
<b>Function descriptions</b>	set auto-refresh interval
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_count</b>	the number SDRAM clock cycles unit between two successive auto-refresh commands, 0x0000~0x1FFF
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the SDRAM auto-refresh rate counter */
exmc_sdram_refresh_count_set(761);
```

### exmc\_sdram\_autorefresh\_number\_set

The description of exmc\_sdram\_autorefresh\_number\_set is shown as below:

**Table 3-347. Function exmc\_sdram\_autorefresh\_number\_set**

<b>Function name</b>	exmc_sdram_autorefresh_number_set
<b>Function prototype</b>	void exmc_sdram_autorefresh_number_set(uint32_t exmc_number);
<b>Function descriptions</b>	set the number of successive auto-refresh command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_number</b>	the number of successive Auto-refresh cycles will be send
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the number of successive auto-refresh command */
exmc_sdram_autorefresh_number_set(10);
```

### exmc\_sdram\_write\_protection\_config

The description of exmc\_sdram\_write\_protection\_config is shown as below:

Table 3-348. Function `exmc_sdram_write_protection_config`

Function name	<code>exmc_sdram_write_protection_config</code>
Function prototype	<code>void exmc_sdram_write_protection_config(uint32_t exmc_sdram_device, ControlStatus newvalue);</code>
Function descriptions	config the write protection function
Precondition	-
The called functions	-
Input parameter{in}	
<code>exmc_sdram_device</code>	specifie the SDRAM device
<code>EXMC_SDRAM_DEVICE</code> Ex	x=0,1
Input parameter{in}	
<code>newvalue</code>	ENABLE or DISABLE
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the write protection function */
```

```
exmc_sdram_write_protection_config(EXMC_SDRAM_DEVICE1, ENABLE);
```

### `exmc_sdram_bankstatus_get`

The description of `exmc_sdram_bankstatus_get` is shown as below:

Table 3-349. Function `exmc_sdram_bankstatus_get`

Function name	<code>exmc_sdram_bankstatus_get</code>
Function prototype	<code>uint32_t exmc_sdram_bankstatus_get(uint32_t exmc_sdram_device);</code>
Function descriptions	get the status of SDRAM device0 or device1
Precondition	-
The called functions	-
Input parameter{in}	
<code>exmc_sdram_device</code>	specifie the SDRAM device
<code>EXMC_SDRAM_DEVICE</code> Ex	x=0,1
Output parameter{out}	
-	-
Return value	
<code>uint32_t</code>	the status of SDRAM device

Example:

```
/* get the status of SDRAM device1 */
```

```
uint32_t status;
```

```
status = exmc_sdram_bankstatus_get (EXMC_SDRAM_DEVICE1);
```

### exmc\_sqpsram\_read\_command\_set

The description of exmc\_sqpsram\_read\_command\_set is shown as below:

**Table 3-350. Function exmc\_sqpsram\_read\_command\_set**

<b>Function name</b>	exmc_sqpsram_read_command_set
<b>Function prototype</b>	void exmc_sqpsram_read_command_set(uint32_t read_command_mode,uint32_t read_wait_cycle,uint32_t read_command_code);
<b>Function descriptions</b>	set the read command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>read_command_mode</b>	configure SPI PSRAM read command mode
EXMC_SQPIPSRAM_READ_MODE_DISABLE	not SPI mode
EXMC_SQPIPSRAM_READ_MODE_SPI	SPI mode
EXMC_SQPIPSRAM_READ_MODE_SQPI	SQPI mode
EXMC_SQPIPSRAM_READ_MODE_QPI	QPI mode
<b>Input parameter{in}</b>	
<b>read_wait_cycle</b>	wait cycle number after address phase,0..15
<b>Input parameter{in}</b>	
<b>read_command_code</b>	read command for AHB read transfer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the SQPIPSRAM read command */
```

```
exmc_sqpsram_read_command_set(EXMC_SQPIPSRAM_READ_MODE_SQPI,1,1);
```

### exmc\_sqpsram\_write\_command\_set

The description of exmc\_sqpsram\_write\_command\_set is shown as below:

**Table 3-351. Function exmc\_sqpsram\_write\_command\_set**

<b>Function name</b>	exmc_sqpsram_write_command_set
----------------------	--------------------------------

<b>Function prototype</b>	void exmc_sqpsram_write_command_set(uint32_t write_command_mode,uint32_t write_wait_cycle,uint32_t write_command_code);
<b>Function descriptions</b>	set the write command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>write_command_mode</b>	configure SPI PSRAM write command mode
<i>EXMC_SQPIPSRAM_WRITE_MODE_DISABLE</i>	not SPI mode
<i>EXMC_SQPIPSRAM_WRITE_MODE_SPI</i>	SPI mode
<i>EXMC_SQPIPSRAM_WRITE_MODE_SQPI</i>	SQPI mode
<i>EXMC_SQPIPSRAM_WRITE_MODE_QPI</i>	QPI mode
<b>Input parameter{in}</b>	
<b>write_wait_cycle</b>	wait cycle number after address phase,0..15
<b>Input parameter{in}</b>	
<b>write_command_code</b>	write command for AHB write transfer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the SQPIPSRAM write command */
```

```
exmc_sqpsram_write_command_set(EXMC_SQPIPSRAM_WRITE_MODE_SQPI,1,1);
```

### exmc\_sqpsram\_read\_id\_command\_send

The description of exmc\_sqpsram\_read\_id\_command\_send is shown as below:

**Table 3-352. Function exmc\_sqpsram\_read\_id\_command\_send**

<b>Function name</b>	exmc_sqpsram_read_id_command_send
<b>Function prototype</b>	void exmc_sqpsram_read_id_command_send(void);
<b>Function descriptions</b>	send SPI read ID command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* send SPI read ID command */
```

```
exmc_sqpsram_read_id_command_send();
```

### exmc\_sqpsram\_write\_cmd\_send

The description of exmc\_sqpsram\_write\_cmd\_send is shown as below:

**Table 3-353. Function exmc\_sqpsram\_write\_cmd\_send**

<b>Function name</b>	exmc_sqpsram_write_cmd_send
<b>Function prototype</b>	void exmc_sqpsram_write_cmd_send(void);
<b>Function descriptions</b>	send SPI special command which does not have address and data phase
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* send SPI special command which does not have address and data phase */
```

```
exmc_sqpsram_write_cmd_send();
```

### exmc\_sqpsram\_low\_id\_get

The description of exmc\_sqpsram\_low\_id\_get is shown as below:

**Table 3-354. Function exmc\_sqpsram\_low\_id\_get**

<b>Function name</b>	exmc_sqpsram_low_id_get
<b>Function prototype</b>	uint32_t exmc_sqpsram_low_id_get(void);
<b>Function descriptions</b>	get the EXMC SPI ID low 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 ID low data
-----------------	-----------------

Example:

```
/* the ID low data */

uint32_t id;

id = exmc_sqpsram_low_id_get();
```

### exmc\_sqpsram\_high\_id\_get

The description of exmc\_sqpsram\_high\_id\_get is shown as below:

**Table 3-355. Function exmc\_sqpsram\_low\_id\_get**

<b>Function name</b>	exmc_sqpsram_high_id_get
<b>Function prototype</b>	uint32_t exmc_sqpsram_high_id_get(void);
<b>Function descriptions</b>	get the EXMC SPI ID high 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 ID high data

Example:

```
/* the ID high data */

uint32_t id;

id = exmc_sqpsram_high_id_get();
```

### exmc\_sqpsram\_send\_command\_state\_get

The description of exmc\_sqpsram\_send\_command\_state\_get is shown as below:

**Table 3-356. Function exmc\_sqpsram\_send\_command\_state\_get**

<b>Function name</b>	exmc_sqpsram_send_command_state_get
<b>Function prototype</b>	FlagStatus exmc_sqpsram_send_command_state_get(uint32_t send_command_flag);
<b>Function descriptions</b>	get the bit value of EXMC send write command bit or read ID command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>send_command_flag</b>	the send command flag
<b>EXMC_SEND_COMMA</b>	EXMC_SRCMD_RDID flag bit

<i>ND_FLAG_RDID</i>	
<i>EXMC_SEND_COMMA</i> <i>ND_FLAG_SC</i>	EXMC_SWCMD_SC flag bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check EXMC_SEND_COMMAND_FLAG_SC is set or not*/
```

```
if(RESET !=exmc_sqpsram_send_command_state_get(EXMC_SEND_COMMAND_FLAG_SC));
```

### exmc\_interrupt\_enable

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

**Table 3-357. Function exmc\_interrupt\_enable**

<b>Function name</b>	exmc_interrupt_enable
<b>Function prototype</b>	void exmc_interrupt_enable(uint32_t exmc_bank,uint32_t interrupt);
<b>Function descriptions</b>	enable EXMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA</i> <i>RD</i>	the PC Card bank
<i>EXMC_SDRAM_DEVIC</i> <i>E0</i>	the SDRAM device0
<i>EXMC_SDRAM_DEVIC</i> <i>E1</i>	the SDRAM device1
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which interrupt flag
<i>EXMC_NAND_PCCAR</i> <i>D_INT_FLAG_RISE</i>	rising edge interrupt and flag
<i>EXMC_NAND_PCCAR</i> <i>D_INT_FLAG_LEVEL</i>	high-level interrupt and flag
<i>EXMC_NAND_PCCAR</i> <i>D_INT_FLAG_FALL</i>	falling edge interrupt and flag
<i>EXMC_SDRAM_INT_F</i> <i>LAG_REFRESH</i>	refresh error interrupt and flag
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* enable EXMC interrupt*/
```

```
exmc_interrupt_enable(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

### exmc\_interrupt\_disable

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

**Table 3-358. Function exmc\_interrupt\_disable**

<b>Function name</b>	exmc_interrupt_disable
<b>Function prototype</b>	void exmc_interrupt_disable(uint32_t exmc_bank, uint32_t interrupt);
<b>Function descriptions</b>	disable EXMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCARD	the PC Card bank
EXMC_SDRAM_DEVICE0	the SDRAM device0
EXMC_SDRAM_DEVICE1	the SDRAM device1
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which interrupt flag
EXMC_NAND_PCCARD_INT_FLAG_RISE	rising edge interrupt and flag
EXMC_NAND_PCCARD_INT_FLAG_LEVEL	high-level interrupt and flag
EXMC_NAND_PCCARD_INT_FLAG_FALL	falling edge interrupt and flag
EXMC_SDRAM_INTERRUPT_FLAG_REFRESH	refresh error interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable EXMC interrupt*/
```

```
exmc_interrupt_disable(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

### exmc\_flag\_get

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

**Table 3-359. Function exmc\_flag\_get**

<b>Function name</b>	exmc_flag_get
<b>Function prototype</b>	FlagStatus exmc_flag_get(uint32_t exmc_bank,uint32_t flag);
<b>Function descriptions</b>	get EXMC flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCA RD	the PC Card bank
EXMC_SDRAM_DEVIC E0	the SDRAM device0
EXMC_SDRAM_DEVIC E1	the SDRAM device1
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
EXMC_NAND_PCCAR D_FLAG_RISE	interrupt rising edge status
EXMC_NAND_PCCAR D_FLAG_LEVEL	interrupt high-level status
EXMC_NAND_PCCAR D_FLAG_FALL	interrupt falling edge status
EXMC_SDRAM_FLAG _REFRESH	refresh error interrupt flag
EXMC_SDRAM_FLAG _NREADY	not ready status
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check EXMC_NAND_PCCARD_FLAG_RISE is set or not*/
```

```
if(RESET != exmc_flag_get(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_FLAG_RISE));
```

### exmc\_flag\_clear

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

**Table 3-360. Function exmc\_flag\_clear**

<b>Function name</b>	exmc_flag_clear
<b>Function prototype</b>	FlagStatus exmc_flag_clear (uint32_t exmc_bank,uint32_t flag);
<b>Function descriptions</b>	clear EXMC flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCARD	the PC Card bank
EXMC_SDRAM_DEVICE0	the SDRAM device0
EXMC_SDRAM_DEVICE1	the SDRAM device1
<b>Input parameter{in}</b>	
<b>flag</b>	specify get which flag
EXMC_NAND_PCCARD_FLAG_RISE	interrupt rising edge status
EXMC_NAND_PCCARD_FLAG_LEVEL	interrupt high-level status
EXMC_NAND_PCCARD_FLAG_FALL	interrupt falling edge status
EXMC_SDRAM_FLAG_REFRESH	refresh error interrupt flag
EXMC_SDRAM_FLAG_NREADY	not ready status
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear EXMC flag status */
exmc_flag_clear(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_FLAG_RISE);
```

## exmc\_interrupt\_flag\_get

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

**Table 3-361. Function exmc\_interrupt\_flag\_get**

<b>Function name</b>	exmc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus exmc_interrupt_flag_get(uint32_t exmc_bank,uint32_t interrupt);
<b>Function descriptions</b>	get EXMC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCARD	the PC Card bank
EXMC_SDRAM_DEVICE0	the SDRAM device0
EXMC_SDRAM_DEVICE1	the SDRAM device1
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which interrupt flag
EXMC_NAND_PCCARD_INT_FLAG_RISE	rising edge interrupt and flag
EXMC_NAND_PCCARD_INT_FLAG_LEVEL	high-level interrupt and flag
EXMC_NAND_PCCARD_INT_FLAG_FALL	falling edge interrupt and flag
EXMC_SDRAM_INTERRUPT_FLAG_REFRESH	refresh error interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check EXMC_NAND_PCCARD_INT_FLAG_RISE is set or not*/
```

```
if(RESET != exmc_interrupt_flag_get (EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_FLAG_RISE));
```

## exmc\_interrupt\_flag\_clear

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

**Table 3-362. Function exmc\_interrupt\_flag\_clear**

<b>Function name</b>	exmc_interrupt_flag_clear
<b>Function prototype</b>	void exmc_interrupt_flag_clear(uint32_t exmc_bank, uint32_t interrupt);
<b>Function descriptions</b>	clear EXMC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exmc_bank</b>	specifies the NAND bank , PC card bank or SDRAM device
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCARD	the PC Card bank
EXMC_SDRAM_DEVICE0	the SDRAM device0
EXMC_SDRAM_DEVICE1	the SDRAM device1
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify get which interrupt flag
EXMC_NAND_PCCARD_INT_FLAG_RISE	rising edge interrupt and flag
EXMC_NAND_PCCARD_INT_FLAG_LEVEL	high-level interrupt and flag
EXMC_NAND_PCCARD_INT_FLAG_FALL	falling edge interrupt and flag
EXMC_SDRAM_INTERRUPT_FLAG_REFRESH	refresh error interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear EXMC interrupt flag */
```

```
exmc_interrupt_flag_clear(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

## 3.12. EXTI

EXTI is the interrupt/event controller in the MCU. It contains up to 23 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.12.1](#), the EXTI firmware functions are introduced in chapter [3.12.2](#)

### 3.12.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

**Table 3-363. 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

### 3.12.2. Descriptions of Peripheral functions

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

**Table 3-364. EXTI firmware function**

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

#### Enum exti\_line\_enum

**Table 3-365. Enum exti\_line\_enum**

Member name	Function description
EXTI_0	EXTI line 0
EXTI_1	EXTI line 1
EXTI_2	EXTI line 2
EXTI_3	EXTI line 3
EXTI_4	EXTI line 4
EXTI_5	EXTI line 5
EXTI_6	EXTI line 6
EXTI_7	EXTI line 7



Member name	Function description
EXTI_8	EXTI line 8
EXTI_9	EXTI line 9
EXTI_10	EXTI line 10
EXTI_11	EXTI line 11
EXTI_12	EXTI line 12
EXTI_13	EXTI line 13
EXTI_14	EXTI line 14
EXTI_15	EXTI line 15
EXTI_16	EXTI line 16
EXTI_17	EXTI line 17
EXTI_18	EXTI line 18
EXTI_19	EXTI line 19
EXTI_20	EXTI line 20
EXTI_21	EXTI line 21
EXTI_22	EXTI line 22

### Enum exti\_mode\_enum

Table 3-366. Enum exti\_mode\_enum

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

### Enum exti\_trig\_type\_enum

Table 3-367. 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 3-368. 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 3-369. Function exti\_init**

<b>Function name</b>	exti_init
<b>Function prototype</b>	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
<b>Function descriptions</b>	initialize EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-365. Enum exti_line_enum</a>
Input parameter{in}	
<b>mode</b>	EXTI mode, refer to <a href="#">Table 3-366. Enum exti_mode_enum</a>
Input parameter{in}	
<b>trig_type</b>	trigger type, refer to <a href="#">Table 3-367. 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 3-370. Function exti\_interrupt\_enable**

<b>Function name</b>	exti_interrupt_enable
<b>Function prototype</b>	void exti_interrupt_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable the interrupts from EXTI line x

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-365. 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 3-371. 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 3-365. 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 3-372. Function exti\_event\_enable**

<b>Function name</b>	exti_event_enable
<b>Function prototype</b>	void exti_event_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable the events from EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-365. 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 3-373. 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 3-365. 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 3-374. 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 3-365. 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 3-375. 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 EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to <a href="#">Table 3-365. 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 3-376. 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 3-365. 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 3-377. 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 3-365. 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 3-378. 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 3-365. 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 3-379. Function exti\_interrupt\_flag\_clear**

<b>Function name</b>	exti_interrupt_flag_clear
<b>Function prototype</b>	void exti_interrupt_flag_clear(exti_line_enum linex);
<b>Function descriptions</b>	clear EXTI line x interrupt pending flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x, refer to <a href="#">Table 3-365. Enum exti_line_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear EXTI line 0 interrupt flag status */
```

```
exti_interrupt_flag_clear(EXTI_0);
```

## 3.13. FMC

There is flash controller and option byte for GD32A490 series. The FMC registers are listed in chapter [3.13.1](#), the FMC firmware functions are introduced in chapter [3.13.2](#).

### 3.13.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

**Table 3-380. FMC Registers**

Registers	Descriptions
FMC_WS	Wait state register
FMC_KEY	Unlock key register
FMC_OBKEY	Option byte unlock key register
FMC_STAT	Status register
FMC_CTL	Control register
FMC_OBCTL0	Option byte control register 0
FMC_OBCTL1	Option byte control register 1

Registers	Descriptions
FMC_PECFG	Page erase configuration register
FMC_PEKEY	Unlock page erase key register
FMC_WSEN	Wait state enable register
FMC_PID	Product ID register

### 3.13.2. Descriptions of Peripheral functions

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

**Table 3-381. FMC firmware function**

Function name	Function description
fmc_wsnt_set	set the FMC wait state counter
fmc_unlock	unlock the main FMC operation
fmc_lock	lock the main FMC operation
fmc_page_erase	FMC erase page
fmc_sector_erase	FMC erase sector
fmc_mass_erase	FMC erase whole chip
fmc_bank0_erase	FMC erase whole bank0
fmc_bank1_erase	FMC erase whole bank1
fmc_word_program	FMC program a word at the corresponding address
fmc_halfword_program	FMC program a half word at the corresponding address
fmc_byte_program	FMC program a byte at the corresponding address
ob_unlock	unlock the option byte operation
ob_lock	lock the option byte operation
ob_start	send option byte change command
ob_erase	erase and reset option byte
ob_write_protection_enable	enable write protect
ob_write_protection_disable	disable write protect
ob_drp_enable	enable erase/program protection and D-bus read protection
ob_drp_disable	disable erase/program protection and D-bus read protection
ob_security_protection_config	set the option byte security protection level
ob_user_write	write the FMC option byte user
ob_user_bor_threshold	option byte BOR threshold value
ob_boot_mode_config	configure the boot mode
ob_user_get	get the FMC option byte user
ob_write_protection0_get	get the FMC option byte write protection
ob_write_protection1_get	get the FMC option byte write protection
ob_drp0_get	get the FMC erase/program protection and D-bus read protection option bytes value
ob_drp1_get	get the FMC erase/program protection and D-bus read protection option bytes value
ob_spc_get	get option byte security protection code value
ob_user_bor_threshold_get	get the FMC threshold value



Function name	Function description
fmc_flag_get	get flag set or reset
fmc_flag_clear	clear the FMC pending flag
fmc_interrupt_enable	enable FMC interrupt
fmc_interrupt_disable	disable FMC interrupt
fmc_interrupt_flag_get	get interrupt flag set or reset
fmc_interrupt_flag_clear	clear the FMC pending interrupt flag
fmc_state_get	return the FMC state
fmc_ready_wait	check FMC ready or not

## fmc\_state\_enum

Table 3-382. fmc\_state\_enum

enum name	enum description
FMC_READY	the operation has been completed
FMC_BUSY	the operation is in progress
FMC_RDERR	read D-bus protection error
FMC_PGSERR	program sequence error
FMC_PGMERR	program size not match error
FMC_WPERR	erase/program protection error
FMC_OPERR	operation error
FMC_TOERR	timeout error

## fmc\_wscnt\_set

The description of fmc\_wscnt\_set is shown as below:

Table 3-383. Function fmc\_wscnt\_set

Function name	fmc_wscnt_set
Function prototype	void fmc_wscnt_set(uint32_t wscnt);
Function descriptions	set the wait state counter value
Precondition	-
The called functions	-
Input parameter{in}	
wscnt	wait state counter value
WS_WSCNT_x	FMC X(X = 0~15) wait
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the wait state counter value */
fmc_wscnt_set (WS_WSCNT_1);
```

## fmc\_unlock

The description of fmc\_unlock is shown as below:

**Table 3-384. Function fmc\_unlock**

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

Example:

```
/* unlock the main FMC operation */
fmc_unlock ( );
```

## fmc\_lock

The description of fmc\_lock is shown as below:

**Table 3-385. Function fmc\_lock**

<b>Function name</b>	fmc_lock
<b>Function prototype</b>	void fmc_lock(void);
<b>Function descriptions</b>	lock the main FMC operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock the main FMC operation */
fmc_lock( );
```

## fmc\_page\_erase

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

Table 3-386. Function fmc\_page\_erase

Function name	fmc_page_erase
Function prototype	fmc_state_enum fmc_page_erase(uint32_t page_addr);
Function descriptions	erase page
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
page_addr	page address
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to <a href="#">fmc_state_enum</a>

Example:

```
/* erase page */

fmc_unlock();

fmc_state_enum state = FMC_READY;

state = fmc_page_erase(0x08008000);
```

### fmc\_sector\_erase

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

Table 3-387. Function fmc\_sector\_erase

Function name	fmc_sector_erase
Function prototype	fmc_state_enum fmc_sector_erase(uint32_t fmc_sector);
Function descriptions	erase sector
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
fmc_sector	select the sector to erase
CTL_SECTOR_NUMB ER_x	sector X(X = 0~27)
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to <a href="#">fmc_state_enum</a>

Example:

```
/* erase number 15 sector */

fmc_unlock();

fmc_state_enum state = FMC_READY;
```

```
state = fmc_sector_erase(CTL_SECTOR_NUMBER_15);
```

### fmc\_mass\_erase

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

**Table 3-388. Function fmc\_mass\_erase**

<b>Function name</b>	fmc_mass_erase
<b>Function prototype</b>	fmc_state_enum fmc_mass_erase(void );
<b>Function descriptions</b>	erase whole chip
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">fmc_state_enum</a>

Example:

```
/* erase the whole chip */

fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_mass_erase();
```

### fmc\_bank0\_erase

The description of fmc\_bank0\_erase is shown as below:

**Table 3-389. 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 all FMC sectors in bank0
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">fmc_state_enum</a>

Example:

```
/* erase bank0 */
```

```
fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_bank0_erase( );
```

### fmc\_bank1\_erase

The description of fmc\_bank1\_erase is shown as below:

**Table 3-390. 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 all FMC sectors in bank0
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, refer to <a href="#">fmc_state_enum</a>

Example:

```
/* erase bank1 */

fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_bank1_erase( );
```

### fmc\_word\_program

The description of fmc\_word\_program is shown as below:

**Table 3-391. Function fmc\_word\_program**

<b>Function name</b>	fmc_word_program
<b>Function prototype</b>	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
<b>Function descriptions</b>	program a word at the corresponding address
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
address	the address to program
data	word to program(0x00000000 - 0xFFFFFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">fmc_state_enum</a>
-----------------------	-------------------------------------------------------

Example:

```
fmc_unlock( );
```

```
fmc_state_enum state = FMC_READY;
```

```
state = fmc_word_program(0x08004000, 0x12345678);
```

## fmc\_halfword\_program

The description of fmc\_halfword\_program is shown as below:

**Table 3-392. Function fmc\_halfword\_program**

<b>Function name</b>	fmc_halfword_program
<b>Function prototype</b>	fmc_state_enum fmc_halfword_program(uint32_t address, uint16_t data);
<b>Function descriptions</b>	program a halfword at the corresponding address
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>address</b>	the address to program
<b>data</b>	halfword to program(0x0000 - 0xFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">fmc_state_enum</a>

Example:

```
/* program half word at the corresponding address */
```

```
fmc_unlock( );
```

```
fmc_state_enum state = FMC_READY;
```

```
state = fmc_halfword_program (0x08004000, 0x1234);
```

## fmc\_byte\_program

The description of fmc\_byte\_program is shown as below:

**Table 3-393. Function fmc\_halfword\_program**

<b>Function name</b>	fmc_halfword_program
<b>Function prototype</b>	fmc_state_enum fmc_byte_program(uint32_t address, uint8_t data);
<b>Function descriptions</b>	program a byte at the corresponding address
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>address</b>	the address to program

<b>data</b>	byte to program(0x00 - 0xFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, refer to <a href="#">fmc_state_enum</a>

Example:

```
/* program a byte at the corresponding address */
```

```
fmc_unlock( );
```

```
fmc_state_enum state = FMC_READY;
```

```
state = fmc_byte_program (0x08004000, 0x12);
```

## ob\_unlock

The description of ob\_unlock is shown as below:

**Table 3-394. 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 3-395. 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>	-

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

Example:

```
/* lock the option byte operation */
```

```
ob_lock( );
```

### ob\_start

The description of ob\_start is shown as below:

**Table 3-396. Function ob\_start**

Function name	ob_start
Function prototype	void ob_start(void);
Function descriptions	send option byte change command
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable write protection and then send the start command */
```

```
ob_unlock( );
```

```
ob_write_protection_enable (OB_WP7);
```

```
ob_start();
```

### ob\_erase

The description of ob\_erase is shown as below:

**Table 3-397. Function ob\_erase**

Function name	ob_erase
Function prototype	void ob_erase(void);
Function descriptions	erase and reset option byte
Precondition	ob_unlock



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

Example:

```
/* erase and reset option byte */
```

```
ob_unlock( );
```

```
ob_erase( );
```

```
ob_start( );
```

```
ob_lock( );
```

### ob\_write\_protection\_enable

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

**Table 3-398. 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>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>ob_wp</b>	specify sector to be write protected
<i>OB_WP_x</i>	sector x(x = 0,1,2...22)
<i>OB_WP_23_27</i>	sector23~27
<i>OB_WP_ALL</i>	all sector
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* enable write protection of sector 7 */
```

```
ErrStatus temp = ERROR;
```

```
ob_unlock( );
```

```
temp = ob_write_protection_enable (OB_WP7);
```

```
ob_start();
```

```
ob_lock( );
```

### ob\_write\_protection\_disable

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

**Table 3-399. 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>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>ob_wp</b>	specify sector to be write protected
<i>OB_WP_x</i>	sector x(x = 0,1,2...22)
<i>OB_WP_23_27</i>	sector23~27
<i>OB_WP_ALL</i>	all sector
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* disable write protection of sector 7 */

ErrStatus temp = ERROR;

ob_unlock( );

temp = ob_write_protection_disable (OB_WP7);

ob_start();

ob_lock( );
```

### ob\_drp\_enable

The description of ob\_drp\_enable is shown as below:

**Table 3-400. Function ob\_drp\_enable**

<b>Function name</b>	ob_drp_enable
<b>Function prototype</b>	void ob_drp_enable(uint32_t ob_drp);
<b>Function descriptions</b>	enable erase/program protection and D-bus read protection
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>ob_drp</b>	enable the WPx bits used as erase/program protection and D-bus read protection of each sector

<i>OB_DRP_x</i>	sector x(x = 0,1,2...22)
<i>OB_DRP_23_27</i>	sector23~27
<i>OB_DRP_ALL</i>	all sector
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable write protection and D-BUS read protectio of sector 7 */
```

```
ob_unlock( );
```

```
ob_drp_enable (OB_DRP7);
```

```
ob_start();
```

```
ob_lock( );
```

### ob\_drp\_disable

The description of ob\_drp\_disable is shown as below:

**Table 3-401. Function ob\_drp\_disable**

<b>Function name</b>	ob_drp_disable
<b>Function prototype</b>	void ob_drp_disable(uint32_t ob_drp);
<b>Function descriptions</b>	disable erase/program protection and D-bus read protection
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>ob_drp</b>	disable the WPx bits used as erase/program protection and D-bus read protection of each sector
<i>OB_DRP_x</i>	sector x(x = 0,1,2...22)
<i>OB_DRP_23_27</i>	sector23~27
<i>OB_DRP_ALL</i>	all sector
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable write protection and D-BUS read protectio of sector 7 */
```

```
ob_unlock( );
```

```
ob_drp_disable (OB_DRP7);
```

```
ob_start();
```

```
ob_lock( );
```

## ob\_security\_protection\_config

The description of ob\_security\_protection\_config is shown as below:

**Table 3-402. Function ob\_security\_protection\_config**

<b>Function name</b>	ob_security_protection_config
<b>Function prototype</b>	void ob_security_protection_config(uint8_t ob_spc);
<b>Function descriptions</b>	configure security protection level
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>ob_spc</b>	specify security protection
<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>	
-	state of FMC, refer to <a href="#">fmc_state_enum</a>

Example:

```
/* config FMC as low security protection */
ob_unlock( );
ob_security_protection_config (FMC_LSPC);
ob_start( );
ob_lock( );
```

## ob\_user\_write

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

**Table 3-403. Function ob\_user\_write**

<b>Function name</b>	ob_user_write
<b>Function prototype</b>	void ob_user_write(uint32_t ob_fwdgt, uint32_t ob_deepsleep, uint32_t ob_stdby);
<b>Function descriptions</b>	program the FMC user option byte
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>ob_fwdgt</b>	option byte watchdog value
<i>OB_FWDGT_SW</i>	software free watchdog

<i>OB_FWDGT_HW</i>	hardware free watchdog
<b>Input parameter{in}</b>	
<b>ob_deepsleep</b>	option byte deepsleep reset value
<i>OB_DEEPSLEEP_NRS</i> <i>T</i>	no reset when entering deepsleep mode
<i>OB_DEEPSLEEP_RST</i>	generate a reset instead of entering deepsleep mode
<b>Input parameter{in}</b>	
<b>ob_stdby</b>	option byte standby reset value
<i>OB_STDBY_NRS</i>	no reset when entering standby mode
<i>OB_STDBY_RST</i>	generate a reset instead of entering standby mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	state of FMC, refer to <a href="#">fmc_state_enum</a>

Example:

```
/* config the USER byte of option byte */
ob_unlock( );

ob_user_write(OB_FWDGT_SW, OB_DEEPSLEEP_NRS, OB_STDBY_NRS);

ob_start ( );

ob_lock( );
```

### ob\_user\_bor\_threshold

The description of ob\_user\_bor\_threshold is shown as below:

**Table 3-404. Function ob\_user\_bor\_threshold**

<b>Function name</b>	ob_user_bor_threshold
<b>Function prototype</b>	void ob_user_bor_threshold(uint32_t ob_bor_th);
<b>Function descriptions</b>	program the option byte BOR threshold value
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ob_bor_th</b>	user option byte
<i>OB_BOR_TH_VALUE3</i>	BOR threshold value 3
<i>OB_BOR_TH_VALUE2</i>	BOR threshold value 2
<i>OB_BOR_TH_VALUE1</i>	BOR threshold value 1
<i>OB_BOR_TH_OFF</i>	no BOR function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config the BOR threshold value as OB_BOR_TH_OFF */
ob_unlock( );
ob_user_bor_threshold(OB_BOR_TH_OFF);
ob_start( );
ob_lock( );
```

### ob\_boot\_mode\_config

The description of ob\_boot\_mode\_config is shown as below:

**Table 3-405. Function ob\_boot\_mode\_config**

<b>Function name</b>	ob_boot_mode_config
<b>Function prototype</b>	void ob_boot_mode_config(uint32_t boot_mode);
<b>Function descriptions</b>	configure the option byte boot bank value
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>boot_mode</b>	specifies the option byte boot bank value
OB_BB_DISABLE	boot from bank0
OB_BB_ENABLE	boot from bank1 or bank0 if bank1 is void
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config boot from bank0 */
ob_unlock( );
ob_boot_mode_config(OB_BB_DISABLE);
ob_start( );
ob_lock( );
```

### ob\_double\_bank\_select

The description of ob\_double\_bank\_select is shown as below:

**Table 3-406. Function ob\_double\_bank\_select**

<b>Function name</b>	ob_double_bank_select
<b>Function prototype</b>	void ob_double_bank_select(uint32_t double_bank);
<b>Function descriptions</b>	configure the option byte double bank select, only for 1MB flash memory

	series
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>double_bank</b>	specifies the option byte double bank select
<i>OB_DBS_DISABLE</i>	single bank when flash size is 1M bytes
<i>OB_DBS_ENABLE</i>	double banks when flash size is 1M bytes
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select single bank */
ob_unlock( );
ob_double_bank_select(OB_DBS_DISABLE);
ob_start( );
ob_lock( );
```

### ob\_user\_get

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

**Table 3-407. Function ob\_user\_get**

<b>Function name</b>	ob_user_get
<b>Function prototype</b>	uint8_t ob_user_get(void);
<b>Function descriptions</b>	get the FMC user option byte
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	the FMC user option byte values: ob_fwdgt(Bit0), ob_deepsleep(Bit1), ob_stdby(Bit2)

Example:

```
/* get the the FMC user option byte values */
uint8_t ob_user_values = 0;
ob_user_values = ob_user_get( );
```

## ob\_write\_protection0\_get

The description of ob\_write\_protection0\_get is shown as below:

**Table 3-408. Function ob\_write\_protection0\_get**

<b>Function name</b>	ob_write_protection0_get
<b>Function prototype</b>	uint16_t ob_write_protection0_get(void);
<b>Function descriptions</b>	get the FMC option byte write protection of bank0
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>Uint16_t</b>	the FMC write protection option byte value

Example:

```
/* get the FMC write protection option byte value */
uint16_t bank0_protection_value = 0;
bank0_protection_value = ob_write_protection0_get( );
```

## ob\_write\_protection1\_get

The description of ob\_write\_protection1\_get is shown as below:

**Table 3-409. Function ob\_write\_protection1\_get**

<b>Function name</b>	ob_write_protection1_get
<b>Function prototype</b>	uint16_t ob_write_protection1_get(void);
<b>Function descriptions</b>	get the FMC option byte write protection of bank1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>Uint16_t</b>	the FMC write protection option byte value

Example:

```
/* get the FMC write protection option byte value */
uint16_t bank1_protection_value = 0;
bank1_protection_value = ob_write_protection1_get( );
```



## ob\_drp0\_get

The description of ob\_drp0\_get is shown as below:

**Table 3-410. Function ob\_drp0\_get**

<b>Function name</b>	ob_drp0_get
<b>Function prototype</b>	uint16_t ob_drp0_get(void);
<b>Function descriptions</b>	get the FMC D-bus read protection protection of bank0
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	the FMC erase/program protection and D-bus read protection option bytes value

Example:

```
/* get the FMC erase/program protection and D-bus read protection option bytes value */
uint16_t bank0_drp_value = 0;
bank0_drp_value = ob_drp0_get( );
```

## ob\_drp1\_get

The description of ob\_drp1\_get is shown as below:

**Table 3-411. Function ob\_drp0\_get**

<b>Function name</b>	ob_drp1_get
<b>Function prototype</b>	uint16_t ob_drp1_get(void);
<b>Function descriptions</b>	get the FMC D-bus read protection protection of bank1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	the FMC erase/program protection and D-bus read protection option bytes value

Example:

```
/* get the FMC erase/program protection and D-bus read protection option bytes value */
uint16_t bank1_drp_value = 0;
```

```
bank1_drp_value = ob_drp1_get( );
```

### ob\_spc\_get

The description of ob\_spc\_get is shown as below:

**Table 3-412. Function ob\_spc\_get**

<b>Function name</b>	ob_spc_get
<b>Function prototype</b>	FlagStatus ob_spc_get(void);
<b>Function descriptions</b>	get the FMC option byte security protection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the FMC option byte security protection */
```

```
FlagStatus spc_value = RESET;
```

```
spc_value = ob_spc_get();
```

### ob\_user\_bor\_threshold\_get

The description of ob\_user\_bor\_threshold\_get is shown as below:

**Table 3-413. Function ob\_user\_bor\_threshold\_get**

<b>Function name</b>	ob_user_bor_threshold_get
<b>Function prototype</b>	uint8_t ob_user_bor_threshold_get(void);
<b>Function descriptions</b>	get the FMC option byte BOR threshold value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	the FMC BOR threshold value:OB_BOR_TH_OFF,OB_BOR_TH_VALUE1,OB_BOR_TH_VALUE2, OB_BOR_TH_VALUE3

Example:

```
/* get the FMC option byte BOR threshold value */
```

```
uint8_t bor_value = 0;
```

```
bor_value = ob_user_bor_threshold_get( );
```

## fmc\_flag\_get

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

**Table 3-414. Function fmc\_flag\_get**

<b>Function name</b>	fmc_flag_get
<b>Function prototype</b>	FlagStatus fmc_flag_get(uint32_t flag);
<b>Function descriptions</b>	check flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	check FMC flag
<i>FMC_FLAG_BUSY</i>	FMC busy flag bit
<i>FMC_FLAG_RDDERR</i>	FMC read D-bus protection error flag bit
<i>FMC_FLAG_PGSERR</i>	FMC program sequence error flag bit
<i>FMC_FLAG_PGMERR</i>	FMC program size not match error flag bit
<i>FMC_FLAG_WPERR</i>	FMC Erase/Program protection error flag bit
<i>FMC_FLAG_OPERR</i>	FMC operation error flag bit
<i>FMC_FLAG_END</i>	FMC end of operation flag bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
FlagStatus flag = RESET;
```

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

## fmc\_flag\_clear

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

**Table 3-415. Function fmc\_flag\_clear**

<b>Function name</b>	fmc_flag_clear
<b>Function prototype</b>	void fmc_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear the FMC flag
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>flag</b>	clear FMC flag
<i>FMC_FLAG_BUSY</i>	FMC busy flag bit
<i>FMC_FLAG_RDDERR</i>	FMC read D-bus protection error flag bit
<i>FMC_FLAG_PGSERR</i>	FMC program sequence error flag bit
<i>FMC_FLAG_PGMERR</i>	FMC program size not match error flag bit
<i>FMC_FLAG_WPERR</i>	FMC Erase/Program protection error flag bit
<i>FMC_FLAG_OPERR</i>	FMC operation error flag bit
<i>FMC_FLAG_END</i>	FMC end of operation flag bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the FMC_FLAG_END flag */
```

```
fmc_flag_clear(FMC_FLAG_END);
```

### fmc\_interrupt\_enable

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

**Table 3-416. Function fmc\_interrupt\_enable**

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

Example:

```
/* enable FMC end of program interrupt */
```

```
fmc_unlock( );
```

```
fmc_interrupt_enable(FMC_INT_END);
```

```
fmc_lock( );
```

## fmc\_interrupt\_disable

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

**Table 3-417. Function fmc\_interrupt\_disable**

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

Example:

```
/* disable FMC interrupt */
fmc_interrupt_disable(FMC_INT_END);
```

## fmc\_interrupt\_flag\_get

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

**Table 3-418. Function fmc\_interrupt\_flag\_get**

<b>Function name</b>	fmc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus fmc_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	check interrupt flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	check FMC flag
<i>FMC_INT_FLAG_RDD_ERR</i>	FMC read D-bus protection error interrupt flag bit
<i>FMC_INT_FLAG_PGS_ERR</i>	FMC program sequence error interrupt flag bit
<i>FMC_INT_FLAG_PGM_ERR</i>	FMC program size not match error interrupt flag bit
<i>FMC_INT_FLAG_WPE_RR</i>	FMC Erase/Program protection error interrupt flag bit
<i>FMC_INT_FLAG_OPE</i>	FMC operation error interrupt flag bit

<i>RR</i>	
<i>FMC_INT_FLAG_END</i>	FMC end of operation interrupt flag bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

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

```
FlagStatus int_flag = RESET;
```

```
int_flag = fmc_interrupt_flag_get(FMC_INT_FLAG_END);
```

### fmc\_interrupt\_flag\_clear

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

**Table 3-419. Function fmc\_interrupt\_flag\_clear**

<b>Function name</b>	fmc_interrupt_flag_clear
<b>Function prototype</b>	void fmc_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear the FMC interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	clear FMC interrupt flag
<i>FMC_INT_FLAG_RDD ERR</i>	FMC read D-bus protection error interrupt flag bit
<i>FMC_INT_FLAG_PGS ERR</i>	FMC program sequence error interrupt flag bit
<i>FMC_INT_FLAG_PGM ERR</i>	FMC program size not match error interrupt flag bit
<i>FMC_INT_FLAG_WPE RR</i>	FMC Erase/Program protection error interrupt flag bit
<i>FMC_INT_FLAG_OPE RR</i>	FMC operation error interrupt flag bit
<i>FMC_INT_FLAG_END</i>	FMC end of operation interrupt flag bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the FMC_INT_FLAG_END flag */
```

```
fmc_interrupt_flag_clear(FMC_INT_FLAG_END);
```

## fmc\_state\_get

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

**Table 3-420. Function fmc\_state\_get**

<b>Function name</b>	fmc_state_get
<b>Function prototype</b>	fmc_state_enum fmc_state_get(void);
<b>Function descriptions</b>	get the FMC state
<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="#">fmc_state_enum</a>

Example:

```
/* get the FMC state */
fmc_state_enum state = fmc_state_get( );
```

## fmc\_ready\_wait

The description of fmc\_ready\_wait is shown as below:

**Table 3-421. Function fmc\_ready\_wait**

<b>Function name</b>	fmc_ready_wait
<b>Function prototype</b>	fmc_state_enum fmc_ready_wait(void);
<b>Function descriptions</b>	check whether FMC is ready or not
<b>Precondition</b>	-
<b>The called functions</b>	fmc_state_get()
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, refer to <a href="#">fmc_state_enum</a>

Example:

```
/* check whether FMC is ready or not */
fmc_state_enum state = fmc_ready_wait ( );
```

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

### 3.14.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

**Table 3-422. FWDGT Registers**

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

### 3.14.2. Descriptions of Peripheral functions

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

**Table 3-423. 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 3-424. 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 3-425. Function fwdgt\_write\_disable

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

Example:

```
/* disable write access to FWDGT_PSC and FWDGT_RLD */
fwdgt_write_disable ( );
```

### fwdgt\_enable

The description of fwdgt\_enable is shown as below:

**Table 3-426. 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 3-427. 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
<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 3-428. 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 3-429. 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 3-430. Function fwdgt\_config**

<b>Function name</b>	fwdgt_config
<b>Function prototype</b>	ErrStatus fwdgt_config(uint16_t reload_value, uint8_t prescaler_div);
<b>Function descriptions</b>	configure counter reload value, and prescaler divider value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>reload_value</b>	specify reload value(0x0000 - 0x0FFF)-
<b>Input parameter{in}</b>	
<b>prescaler_div</b>	FWDGT prescaler value-
<i>FWDGT_PSC_DIV4</i>	FWDGT prescaler set to 4
<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16

<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32
<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64
<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* confiure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
```

```
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

### fwdgt\_flag\_get

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

**Table 3-431. 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
{
...
}

```

## 3.15. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.15.1](#), the GPIO firmware functions are introduced in chapter [错误!未找到引用源。](#).

### 3.15.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

**Table 3-432. GPIO Registers**

Registers	Descriptions
GPIOx_CTL	GPIO port control register
GPIO_OMODE	GPIO port output mode register
GPIO_OSPD	GPIO port output speed register
GPIO_PUD	GPIO port pull-up/pull-down register
GPIO_ISTAT	GPIO port input status register
GPIO_OCTL	GPIO port output control register
GPIO_BOP	GPIO port bit operate register
GPIO_LOCK	GPIO port configuration lock register
GPIO_AFSEL0	GPIO alternate function selected register 0
GPIO_AFSEL1	GPIO alternate function selected register 1
GPIO_BC	GPIO bit clear register
GPIO_TG	GPIO port bit toggle register

### 3.15.2. Descriptions of Peripheral functions

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

**Table 3-433. GPIO firmware function**

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

### gpio\_deinit

The description of gpio\_deinit is shown as below:

**Table 3-434. Function gpio\_deinit**

Function name	gpio_deinit
Function prototype	void gpio_deinit(uint32_t gpio_periph);
Function descriptions	reset GPIO port
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset GPIOA */
gpio_deinit (GPIOA);
```

### gpio\_mode\_set

The description of gpio\_mode\_set is shown as below:

Table 3-435. Function gpio\_mode\_set

Function name	gpio_mode_set
Function prototype	void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin);
Function descriptions	set GPIO mode
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
mode	GPIO pin mode
GPIO_MODE_INPUT	input mode
GPIO_MODE_OUTPUT	output mode
GPIO_MODE_AF	alternate function mode
GPIO_MODE_ANALOG	analog mode
Input parameter{in}	
pull_up_down	GPIO pin with pull-up or pull-down resistor
GPIO_PUPD_NONE	floating mode, no pull-up and pull-down resistors
GPIO_PUPD_PULLUP	with pull-up resistor
GPIO_PUPD_PULLDOWN	with pull-down resistor
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set PA0 with pull-up input mode */
```

```
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 3-436. Function gpio\_output\_options\_set

Function name	gpio_output_options_set
Function prototype	void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);
Function descriptions	set GPIO output type and speed
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
otype	GPIO pin output mode
GPIO_OTYPE_PP	push pull mode
GPIO_OTYPE_OD	open drain mode
Input parameter{in}	
speed	gpio output max speed value
GPIO_OSPEED_2MHZ	output max speed 2MHz
GPIO_OSPEED_25MHZ	output max speed 25MHz
GPIO_OSPEED_50MHZ	output max speed 50MHz
GPIO_OSPEED_MAX	output max speed more than 50MHz
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure PA0 as push pull mode */
```

```
gpio_output_options_set (GPIOA, GPIO_OTYPE_PP, GPIO_OSPEED_2MHZ,
GPIO_PIN_0);
```

### gpio\_bit\_set

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

Table 3-437. Function gpio\_bit\_set

Function name	gpio_bit_set
Function prototype	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,F,G,H,I)
<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 3-438. Function gpio\_bit\_reset**

<b>Function name</b>	gpio_bit_reset
<b>Function prototype</b>	void gpio_bit_reset(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	reset GPIO pin bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
<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 3-439. Function gpio\_bit\_write**

<b>Function name</b>	gpio_bit_write
<b>Function prototype</b>	void gpio_bit_write(uint32_t gpio_periph, uint32_t pin, bit_status bit_value);
<b>Function descriptions</b>	write data to the specified GPIO pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
<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 3-440. 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,F,G,H,I)

Input parameter{in}	
<b>data</b>	specify the value to be written to the port output data register
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

### gpio\_input\_bit\_get

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

**Table 3-441. Function gpio\_input\_bit\_get**

<b>Function name</b>	gpio_input_bit_get
<b>Function prototype</b>	FlagStatus gpio_input_bit_get(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	get GPIO pin input status
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x (x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get status of PA0 */
```

```
FlagStatus bit_state;
```

```
bit_state = gpio_input_bit_get (GPIOA, GPIO_PIN_0);
```

### gpio\_input\_port\_get

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

**Table 3-442. Function gpio\_input\_port\_get**

<b>Function name</b>	gpio_input_port_get
----------------------	---------------------

<b>Function prototype</b>	uint16_t gpio_input_port_get(uint32_t gpio_periph);
<b>Function descriptions</b>	get GPIO port input status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
<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 3-443. 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,F,G,H,I)
<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 or RESET

Example:

```
/* get output status of PA0 */

FlagStatus bit_state;
```

```
bit_state = gpio_output_bit_get (GPIOA, GPIO_PIN_0);
```

## gpio\_output\_port\_get

The description of gpio\_output\_port\_get is shown as below:

**Table 3-444. 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 port 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,F,G,H,I)
<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 3-445. Function gpio\_pin\_remap\_config**

<b>Function name</b>	gpio_af_set
<b>Function prototype</b>	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);
<b>Function descriptions</b>	set GPIO alternate function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
<b>Input parameter{in}</b>	
<b>alt_func_num</b>	GPIO pin af function
<i>GPIO_AF_0</i>	SYSTEM
<i>GPIO_AF_1</i>	TIMER0, TIMER1
<i>GPIO_AF_2</i>	TIMER2, TIMER3, TIMER4
<i>GPIO_AF_3</i>	TIMER7, TIMER8, TIMER9, TIMER10

<i>GPIO_AF_4</i>	I2C0, I2C1, I2C2
<i>GPIO_AF_5</i>	SPI0, SPI1, SPI2, SPI3, SPI4, SPI5
<i>GPIO_AF_6</i>	SPI2, SPI3, SPI4
<i>GPIO_AF_7</i>	USART0, USART1, USART2, SPI1, SPI2
<i>GPIO_AF_8</i>	UART3, UART4, USART5, UART6, UART7
<i>GPIO_AF_9</i>	CAN0, CAN1, TLI, TIMER11, TIMER12, TIMER13, I2C1, I2C2, CTC
<i>GPIO_AF_10</i>	USB_FS, USB_HS
<i>GPIO_AF_11</i>	ENET
<i>GPIO_AF_12</i>	EXMC, SDIO, USB_HS
<i>GPIO_AF_13</i>	DCI
<i>GPIO_AF_14</i>	TLI
<i>GPIO_AF_15</i>	EVENTOUT
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x (x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*set PA0 alternate function 0*/
```

```
gpio_af_set(GPIOA, GPIO_AF_0, GPIO_PIN_0);
```

### gpio\_pin\_lock

The description of gpio\_pin\_lock is shown as below:

**Table 3-446. Function gpio\_pin\_lock**

<b>Function name</b>	gpio_pin_lock
<b>Function prototype</b>	void gpio_pin_lock(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	lock GPIO pin bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x (x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* lock PA0 */
gpio_pin_lock (GPIOA, GPIO_PIN_0);
```

## gpio\_bit\_toggle

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

**Table 3-447. Function gpio\_bit\_toggle**

<b>Function name</b>	gpio_bit_toggle
<b>Function prototype</b>	void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	toggle GPIO pin status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
<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:

```
/* toggle PA0 */
gpio_bit_toggle (GPIOA, GPIO_PIN_0);
```

## gpio\_port\_toggle

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

**Table 3-448. Function gpio\_port\_toggle**

<b>Function name</b>	gpio_port_toggle
<b>Function prototype</b>	void gpio_port_toggle(uint32_t gpio_periph);
<b>Function descriptions</b>	toggle GPIO port status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	



<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
<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:

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

```
gpio_port_toggle (GPIOA);
```

## 3.16. I2C

The I2C (inter-integrated circuit) module provides an I2C interface which is an industry standard two-line serial interface for MCU to communicate with external I2C interface. The I2C registers are listed in chapter [3.16.1](#), the I2C firmware functions are introduced in chapter [3.16.2](#).

### 3.16.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

**Table 3-449. I2C Registers**

Registers	Descriptions
I2C_CTL0	Control register 0
I2C_CTL1	Control register 1
I2C_SADDR0	Slave address register 0
I2C_SADDR1	Slave address register 1
I2C_DATA	Transfer buffer register
I2C_STAT0	Transfer status register 0
I2C_STAT1	Transfer status register 1
I2C_CKCFG	Clock configure register
I2C_RT	Rise time register

Registers	Descriptions
I2C_FCTL	Filter control register
I2C_SAMCS	SAM control and status register

### 3.16.2. Descriptions of Peripheral functions

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

**Table 3-450. I2C firmware function**

Function name	Function description
i2c_deinit	reset I2C
i2c_clock_config	configure I2C clock
i2c_mode_addr_config	configure I2C address
i2c_smbus_type_config	SMBus type selection
i2c_ack_config	whether or not to send an ACK
i2c_ackpos_config	configure I2C POAP position
i2c_master_addressing	master send slave address
i2c_dualaddr_enable	enable dual-address mode
i2c_dualaddr_disable	disable dual-address 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 function
i2c_data_receive	I2C receive data function
i2c_dma_config	configure I2C DMA mode
i2c_dma_last_transfer_config	configure whether next DMA EOT is DMA last transfer or not
i2c_stretch_scl_low_config	whether to stretch SCL low when data is not ready in slave mode
i2c_slave_response_to_gcall_config	whether or not to response to a general call
i2c_software_reset_config	configure software reset of I2C

Function name	Function description
i2c_pec_config	configure I2C PEC calculation
i2c_pec_transfer_config	configure whether to transfer PEC value
i2c_pec_value_get	packet error checking value
i2c_smbus_alert_config	configure I2C alert through SMBA pin
i2c_smbus_arp_config	configure I2C ARP protocol in SMBus
i2c_analog_noise_filter_disable	I2C analog noise filter disable
i2c_analog_noise_filter_enable	I2C analog noise filter enable
i2c_digital_noise_filter_config	digital noise filter
i2c_sam_enable	enable SAM_V interface
i2c_sam_disable	disable SAM_V interface
i2c_sam_timeout_enable	enable SAM_V interface timeout detect
i2c_sam_timeout_disable	disable SAM_V interface timeout detect
i2c_flag_get	get I2C flag status
i2c_flag_clear	clear I2C flag status
i2c_interrupt_enable	enable I2C interrupt
i2c_interrupt_disable	disable I2C interrupt
i2c_interrupt_flag_get	get I2C interrupt flag status
i2c_interrupt_flag_clear	clear I2C interrupt flag status

## i2c\_deinit

The description of i2c\_deinit is shown as below:

**Table 3-451. Function i2c\_deinit**

Function name	i2c_deinit
Function prototype	void i2c_deinit(uint32_t i2c_periph);
Function descriptions	reset I2C
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	

<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
i2c_deinit (I2C0);
```

### i2c\_clock\_config

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

**Table 3-452. Function i2c\_clock\_config**

<b>Function name</b>	i2c_clock_config
<b>Function prototype</b>	void i2c_clock_config(uint32_t i2c_periph, uint32_t clkspeed, uint32_t dutycyc);
<b>Function descriptions</b>	I2C clock configure
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>clkspeed</b>	i2c clock speed
<b>Input parameter{in}</b>	
<b>dutycyc</b>	duty cycle in fast mode
<i>I2C_DTCY_2</i>	T_low/T_high=2
<i>I2C_DTCY_16_9</i>	T_low/T_high=16/9
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure I2C0 clock speed as 100KHz*/
```

```
i2c_clock_config(I2C0, 100000, I2C_DTCY_2);
```

### i2c\_mode\_addr\_config

The description of i2c\_mode\_addr\_config is shown as below:

**Table 3-453. Function i2c\_mode\_addr\_config**

Function name	i2c_mode_addr_config
Function prototype	void i2c_mode_addr_config(uint32_t i2c_periph, uint32_t mode, uint32_t addformat, uint32_t addr);
Function descriptions	configure I2C address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
mode	I2C mode select
I2C_I2CMODE_ENABLER	I2C mode
I2C_SMBUSMODE_ENABLED	SMBus mode
Input parameter{in}	
addformat	7bits or 10bits
I2C_ADDFORMAT_7BITS	address format is 7 bits
I2C_ADDFORMAT_10BITS	address format is 10 bits
Input parameter{in}	

<b>addr</b>	I2C address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2C0 address as 0x82, using 7 bits */
```

```
i2c_mode_addr_config(I2C0, I2C_I2CMODE_ENABLE, I2C_ADDFORMAT_7BITS, 0x82);
```

### i2c\_smbus\_type\_config

The description of i2c\_smbus\_type\_config is shown as below:

**Table 3-454. Function i2c\_smbus\_type\_config**

<b>Function name</b>	i2c_smbus_type_config
<b>Function prototype</b>	void i2c_smbus_type_config(uint32_t i2c_periph, uint32_t type);
<b>Function descriptions</b>	SMBus type selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>type</b>	Device or host
<i>I2C_SMBUS_DEVICE</i>	device
<i>I2C_SMBUS_HOST</i>	host
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config I2C0 as SMBUS host type*/
```

```
i2c_smbus_type_config (I2C0, I2C_SMBUS_HOST);
```

### i2c\_ack\_config

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

**Table 3-455. Function i2c\_ack\_config**

<b>Function name</b>	i2c_ack_config
<b>Function prototype</b>	void i2c_ack_config(uint32_t i2c_periph, uint32_t ack);
<b>Function descriptions</b>	whether or not to send an ACK
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>ack</b>	I2C peripheral
<i>I2C_ACK_ENABLE</i>	ACK will be sent
<i>I2C_ACK_DISABLE</i>	ACK will not be sent
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 will send ACK */
```

```
i2c_ack_config (I2C0, I2C_ACK_ENABLE);
```

### i2c\_ackpos\_config

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

**Table 3-456. Function i2c\_ackpos\_config**

<b>Function name</b>	i2c_ackpos_config
<b>Function prototype</b>	void i2c_ackpos_config(uint32_t i2c_periph, uint32_t pos);

<b>Function descriptions</b>	I2C POAP position configure
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>pos</b>	ACK position
<i>I2C_ACKPOS_CURRENT</i>	whether to send ACK or not for the current
<i>I2C_ACKPOS_NEXT</i>	whether to send ACK or not for the next byte
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

/\* The ACK of I2C0 is send for the current frame\*/

```
i2c_ackpos_config (I2C0, I2C_ACKPOS_CURRENT);
```

## i2c\_master\_addressing

The description of i2c\_master\_addressing is shown as below:

**Table 3-457. Function i2c\_master\_addressing**

<b>Function name</b>	i2c_master_addressing
<b>Function prototype</b>	void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection);
<b>Function descriptions</b>	void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection);
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	



<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>addr</b>	slave address
<b>Input parameter{in}</b>	
<b>trandirection</b>	transmitter or receiver
<i>I2C_TRANSMITTER</i>	transmitter
<i>I2C_RECEIVER</i>	receiver
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* send slave address to I2C bus and I2C0 act as receiver */
```

```
i2c_master_addressing(I2C0, 0x82, I2C_RECEIVER);
```

### i2c\_dualaddr\_enable

The description of i2c\_dualaddr\_enable is shown as below:

**Table 3-458. Function i2c\_dualaddr\_enable**

<b>Function name</b>	i2c_dualaddr_enable
<b>Function prototype</b>	void i2c_dualaddr_enable(uint32_t i2c_periph, uint32_t dualaddr);
<b>Function descriptions</b>	enable dual-address mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>dualaddr</b>	I2C address

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 dual-address*/
```

```
i2c_dualaddr_enable (I2C0, 0x84);
```

### i2c\_dualaddr\_disable

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

**Table 3-459. Function i2c\_dualaddr\_disable**

Function name	i2c_dualaddr_disable
Function prototype	void i2c_dualaddr_disable(uint32_t i2c_periph);
Function descriptions	disable dual-address mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 dual-address*/
```

```
i2c_dualaddr_disable (I2C0);
```

### i2c\_enable

The description of i2c\_enable is shown as below:

Table 3-460. Function i2c\_enable

Function name	i2c_enable
Function prototype	void i2c_enable(uint32_t i2c_periph);
Function descriptions	enable I2C
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

### i2c\_disable

The description of i2c\_disable is shown as below:

Table 3-461. Function i2c\_disable

Function name	i2c_disable
Function prototype	void i2c_disable(uint32_t i2c_periph);
Function descriptions	disable I2C
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable I2C0 */
```

```
i2c_disable (I2C0);
```

### i2c\_start\_on\_bus

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

**Table 3-462. Function i2c\_start\_on\_bus**

Function name	i2c_start_on_bus
Function prototype	void i2c_start_on_bus(uint32_t i2c_periph);
Function descriptions	generate a START condition on I2C bus
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
i2c_start_on_bus (I2C0);
```

### i2c\_stop\_on\_bus

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

**Table 3-463. Function i2c\_stop\_on\_bus**

Function name	i2c_stop_on_bus
Function prototype	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=0,1,2)
<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 3-464. Function i2c\_data\_transmit**

<b>Function name</b>	i2c_data_transmit
<b>Function prototype</b>	void i2c_data_transmit(uint32_t i2c_periph, uint8_t data);
<b>Function descriptions</b>	I2C transmit data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>data</b>	transmit data
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* I2C0 transmit data */
```

```
i2c_data_transmit (I2C0);
```

### i2c\_data\_receive

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

**Table 3-465. Function i2c\_data\_receive**

Function name	i2c_data_receive
Function prototype	uint8_t i2c_data_receive(uint32_t i2c_periph);
Function descriptions	I2C receive data function
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
uint8_t	0x00..0xFF

Example:

```
/* I2C0 receive data */
```

```
uint8_t i2c_receiver;
```

```
i2c_receiver = i2c_data_receive(I2C0);
```

### i2c\_dma\_config

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

**Table 3-466. Function i2c\_dma\_config**

Function name	i2c_dma_config
---------------	----------------

<b>Function prototype</b>	void i2c_dma_config(uint32_t i2c_periph, uint32_t dmastate);
<b>Function descriptions</b>	configure I2C DMA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>dmastate</b>	On or off
<i>I2C_DMA_ON</i>	DMA mode enable
<i>I2C_DMA_OFF</i>	DMA mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 DMA mode enable */
```

```
i2c_dma_config(I2C0, I2C_DMA_ON);
```

### **i2c\_dma\_last\_transfer\_enable**

The description of i2c\_dma\_last\_transfer\_enable is shown as below:

**Table 3-467. Function i2c\_dma\_last\_transfer\_enable**

<b>Function name</b>	i2c_dma_last_transfer_enable
<b>Function prototype</b>	void i2c_dma_last_transfer_enable(uint32_t i2c_periph, uint32_t dmalast);
<b>Function descriptions</b>	flag indicating DMA last transfer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral

<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>dmalast</b>	next DMA EOT is the last transfer or not
<i>I2C_DMALST_ON</i>	next DMA EOT is the last transfer
<i>I2C_DMALST_OFF</i>	next DMA EOT is not the last transfer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* next DMA EOT is the last transfer */
```

```
i2c_dma_last_transfer_enable (I2C0, I2C_DMALST_ON);
```

### i2c\_stretch\_scl\_low\_config

The description of i2c\_stretch\_scl\_low\_config is shown as below:

**Table 3-468. Function i2c\_stretch\_scl\_low\_config**

<b>Function name</b>	i2c_stretch_scl_low_config
<b>Function prototype</b>	void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara);
<b>Function descriptions</b>	whether to stretch SCL low when data is not ready in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>stretchpara</b>	SCL stretching enable or disable
<i>I2C_SCLSTRETCH_ENABLE</i>	enable SCL stretching
<i>I2C_SCLSTRETCH_DISABLE</i>	enable SCL stretching



Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_config (I2C0, I2C_SCLSTRETCH_ENABLE);
```

### i2c\_slave\_response\_to\_gcall\_config

The description of i2c\_slave\_response\_to\_gcall\_config is shown as below:

**Table 3-469. Function i2c\_slave\_response\_to\_gcall\_config**

Function name	i2c_slave_response_to_gcall_config
Function prototype	void i2c_slave_response_to_gcall_config(uint32_t i2c_periph, uint32_t gcallpara);
Function descriptions	whether or not to response to a general call
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
gcallpara	response to a general call or not
I2C_GCEN_ENABLE	slave will response to a general call
I2C_GCEN_DISABLE	slave will not response to a general call
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 will response to a general call */
```

i2c\_slave\_response\_to\_gcall\_config (I2C0, I2C\_GCEN\_ENABLE);

### i2c\_software\_reset\_config

The description of i2c\_software\_reset\_config is shown as below:

**Table 3-470. Function i2c\_software\_reset\_config**

<b>Function name</b>	i2c_software_reset_config
<b>Function prototype</b>	void i2c_software_reset_config(uint32_t i2c_periph, uint32_t sreset);
<b>Function descriptions</b>	configure software reset of I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>sreset</b>	under reset or not
<i>I2C_SRESET_SET</i>	I2C is under reset
<i>I2C_SRESET_RESET</i>	I2C is not under reset
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
i2c_software_reset_config (I2C0, I2C_SRESET_SET);
```

### i2c\_pec\_config

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

**Table 3-471. Function i2c\_pec\_config**

<b>Function name</b>	i2c_pec_config
<b>Function prototype</b>	void i2c_pec_config (uint32_t i2c_periph, uint32_t pecstate);

<b>Function descriptions</b>	configure I2C PEC calculation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>pecstate</b>	On or off
<i>I2C_PEC_ENABLE</i>	PEC calculation on
<i>I2C_PEC_DISABLE</i>	PEC calculation off
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* Enable I2C PEC calculation */
```

```
i2c_pec_config (I2C0, I2C_PEC_ENABLE);
```

### **i2c\_pec\_transfer\_config**

The description of i2c\_pec\_transfer\_config is shown as below:

**Table 3-472. Function i2c\_pec\_transfer\_config**

<b>Function name</b>	i2c_pec_transfer_config
<b>Function prototype</b>	void i2c_pec_transfer_config(uint32_t i2c_periph, uint32_t pecpara);
<b>Function descriptions</b>	configure whether to transfer PEC value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)

Input parameter{in}	
<b>pecpara</b>	Transfer PEC or not
<i>I2C_PECTRANS_ENA</i> <i>BLE</i>	transfer PEC
<i>I2C_PECTRANS_DISA</i> <i>BLE</i>	not transfer PEC
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 transfer PEC */
```

```
i2c_pec_transfer_config(I2C0, I2C_PECTRANS_ENABLE);
```

### i2c\_pec\_value\_get

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

**Table 3-473. Function i2c\_pec\_value\_get**

<b>Function name</b>	i2c_pec_value_get
<b>Function prototype</b>	uint8_t i2c_pec_value_get(uint32_t i2c_periph);
<b>Function descriptions</b>	get packet error checking value
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
<b>uint8_t</b>	0..255

Example:

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

```
uint8_t pec_value;
```

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

### i2c\_smbus\_alert\_config

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

**Table 3-474. Function i2c\_smbus\_alert\_config**

<b>Function name</b>	i2c_smbus_alert_config
<b>Function prototype</b>	void i2c_smbus_alert_config(uint32_t i2c_periph, uint32_t smbuspara);
<b>Function descriptions</b>	configure I2C alert through SMBA pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Input parameter{in}</b>	
<b>smbuspara</b>	issue alert through SMBA pin or not
<i>I2C_SALTSEND_ENABLE</i>	issue alert through SMBA pin
<i>I2C_SALTSEND_DISABLE</i>	not issue alert through SMBA pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 issue alert through SMBA pin enable*/
```

```
i2c_smbus_alert_config(I2C0, I2C_SALTSEND_ENABLE);
```

### i2c\_smbus\_arp\_config

The description of i2c\_smbus\_arp\_config is shown as below:

Table 3-475. Function i2c\_smbus\_arp\_config

Function name	i2c_smbus_arp_config
Function prototype	void i2c_smbus_arp_config(uint32_t i2c_periph, uint32_t arpstate);
Function descriptions	configure I2C ARP protocol in SMBus
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
arpstate	ARP protocol in SMBus switch
I2C_ARP_ENABLE	enable ARP
I2C_ARP_DISABLE	disable ARP
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 ARP protocol in SMBus switch */
i2c_smbus_arp_config(I2C0, I2C_ARP_ENABLE);
```

### i2c\_analog\_noise\_filter\_disable

The description of i2c\_analog\_noise\_filter\_disable is shown as below:

Table 3-476. 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	I2C analog noise filter disable
Precondition	-
The called functions	-
Input parameter{in}	

<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 analog noise filter */
```

```
i2c_analog_noise_filter_disable (I2C0);
```

### **i2c\_analog\_noise\_filter\_enable**

The description of i2c\_analog\_noise\_filter\_enable is shown as below:

**Table 3-477. Function i2c\_analog\_noise\_filter\_enable**

<b>Function name</b>	i2c_analog_noise_filter_enable
<b>Function prototype</b>	void i2c_analog_noise_filter_enable(uint32_t i2c_periph);
<b>Function descriptions</b>	I2C analog noise filter enable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 analog noise filter */
```

```
i2c_analog_noise_filter_enable (I2C0);
```

## i2c\_digital\_noise\_filter\_config

The description of i2c\_digital\_noise\_filter\_config is shown as below:

**Table 3-478. 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, i2c_digital_filter_enum dfilterpara);
<b>Function descriptions</b>	config 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=0,1,2)
<b>Input parameter{in}</b>	
<b>dfilterpara</b>	filtered spiker's length
<i>i2c_digital_filter_enum</i>	maximum filter spikes's length
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config I2C0 digital noise filter as I2C_DF_1PCLK */
i2c_digital_noise_filter_config (I2C0, I2C_DF_1PCLK);
```

## i2c\_sam\_enable

The description of i2c\_sam\_enable is shown as below:

**Table 3-479. Function i2c\_sam\_enable**

<b>Function name</b>	i2c_sam_enable
<b>Function prototype</b>	void i2c_sam_enable (uint32_t i2c_periph);
<b>Function descriptions</b>	enable SAM_V interface
<b>Precondition</b>	-



<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 SAM_V interface */
```

```
i2c_sam_enable (I2C0);
```

### **i2c\_sam\_disable**

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

**Table 3-480. Function i2c\_sam\_disable**

<b>Function name</b>	i2c_sam_disable
<b>Function prototype</b>	void i2c_sam_disable (uint32_t i2c_periph);
<b>Function descriptions</b>	disable SAM_V interface
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 SAM_V interface*/
```

```
i2c_sam_disable (I2C0);
```

## i2c\_sam\_timeout\_enable

The description of i2c\_sam\_timeout\_enable is shown as below:

**Table 3-481. Function i2c\_sam\_timeout\_enable**

<b>Function name</b>	i2c_sam_timeout_enable
<b>Function prototype</b>	void i2c_sam_timeout_enable (uint32_t i2c_periph);
<b>Function descriptions</b>	enable SAM_V interface timeout detect
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 SAM_V interface timeout detect */
i2c_sam_timeout_enable (I2C0);
```

## i2c\_sam\_timeout\_disable

The description of i2c\_sam\_timeout\_disable is shown as below:

**Table 3-482. Function i2c\_sam\_timeout\_disable**

<b>Function name</b>	i2c_sam_timeout_disable
<b>Function prototype</b>	void i2c_sam_timeout_disable (uint32_t i2c_periph);
<b>Function descriptions</b>	disable SAM_V interface timeout detect
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 SAM_V interface timeout detect */
```

```
i2c_sam_timeout_disable (I2C0);
```

### i2c\_flag\_get

The description of i2c\_flag\_get is shown as below:

**Table 3-483. Function i2c\_flag\_get**

Function name	i2c_flag_get
Function prototype	FlagStatus i2c_flag_get(uint32_t i2c_periph,uint32_t flag );
Function descriptions	get I2C flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
flag	specify get which flag
I2C_FLAG_SBSEND	start condition send out
I2C_FLAG_ADDSEND	address is sent in master mode or received and matches in slave mode
I2C_FLAG_BTC	byte transmission finishes
I2C_FLAG_ADD10SEND	header of 10-bit address is sent in master mode
I2C_FLAG_STPDET	stop condition detected in slave mode
I2C_FLAG_RBNE	I2C_DATA is not Empty during receiving
I2C_FLAG_TBE	I2C_DATA is empty during transmitting
I2C_FLAG_BERR	a bus error occurs indication a unexpected start or stop condition on I2C

	bus
<i>I2C_FLAG_LOSTARB</i>	arbitration lost in master mode
<i>I2C_FLAG_AERR</i>	acknowledge error
<i>I2C_FLAG_OUERR</i>	overflow or underrun situation occurs in slave mode
<i>I2C_FLAG_PECERR</i>	PEC error when receiving data
<i>I2C_FLAG_SMBTO</i>	timeout signal in SMBus mode
<i>I2C_FLAG_SMBALT</i>	SMBus alert status
<i>I2C_FLAG_MASTER</i>	a flag indicating whether I2C block is in master or slave mode
<i>I2C_FLAG_I2CBSY</i>	busy flag
<i>I2C_FLAG_TRS</i>	whether the I2C is a transmitter or a receiver
<i>I2C_FLAG_RXGC</i>	general call address (00h) received
<i>I2C_FLAG_DEFSMB</i>	default address of SMBus device
<i>I2C_FLAG_HSTSMB</i>	SMBus host header detected in slave mode
<i>I2C_FLAG_DUMOD</i>	dual flag in slave mode indicating which address is matched in dual-address mode
<i>I2C_FLAG_TFF</i>	txframe fall flag
<i>I2C_FLAG_TFR</i>	txframe rise flag
<i>I2C_FLAG_RFF</i>	rxframe fall flag
<i>I2C_FLAG_RFR</i>	rxframe rise flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / RESET

Example:

```
/* check whether start condition send out */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get (I2C0, I2C_FLAG_SBSEND);
```

### **i2c\_flag\_clear**

The description of i2c\_flag\_clear is shown as below:

Table 3-484. Function i2c\_flag\_clear

Function name	i2c_flag_clear
Function prototype	void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);
Function descriptions	clear I2C flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
flag	flag type
I2C_FLAG_SMBALT	SMBus Alert status
I2C_FLAG_SMBTO	timeout signal in SMBus mode
I2C_FLAG_PECERR	PEC error when receiving data
I2C_FLAG_OUERR	over-run or under-run situation occurs in slave mode
I2C_FLAG_AERR	acknowledge error
I2C_FLAG_LOSTARB	arbitration lost in master mode
I2C_FLAG_BERR	a bus error
I2C_FLAG_ADDSEND	cleared by reading I2C_STAT0 and reading I2C_STAT1
I2C_FLAG_TFF	txframe fall flag
I2C_FLAG_TFR	txframe rise flag
I2C_FLAG_RFF	rxframe fall flag
I2C_FLAG_RFR	rxframe rise flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear a bus error flag*/
```

```
i2c_flag_clear (I2C0, I2C_FLAG_BERR);
```

## i2c\_interrupt\_enable

The description of i2c\_interrupt\_enable is shown as below:

**Table 3-485. 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=0,1,2)
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type
<i>I2C_INT_ERR</i>	error interrupt enable
<i>I2C_INT_EV</i>	event interrupt enable
<i>I2C_INT_BUF</i>	buffer interrupt enable
<i>I2C_INT_TFF</i>	txframe fall interrupt enable
<i>I2C_INT_TFR</i>	txframe rise interrupt enable
<i>I2C_INT_RFF</i>	rxframe fall interrupt enable
<i>I2C_INT_RFR</i>	rxframe rise interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 error interrupt */
```

```
i2c_interrupt_enable (I2C0, I2C_INT_EV);
```

## i2c\_interrupt\_disable

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

**Table 3-486. 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=0,1,2)
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type
<i>I2C_INT_ERR</i>	error interrupt disable
<i>I2C_INT_EV</i>	event interrupt disable
<i>I2C_INT_BUF</i>	buffer interrupt disable
<i>I2C_INT_TFF</i>	<i>txframe fall interrupt enable</i>
<i>I2C_INT_TFR</i>	<i>txframe rise interrupt enable</i>
<i>I2C_INT_RFF</i>	<i>rxframe fall interrupt enable</i>
<i>I2C_INT_RFR</i>	<i>rxframe rise interrupt enable</i>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 error interrupt */
i2c_interrupt_disable (I2C0, I2C_INT_EV);
```

## i2c\_interrupt\_flag\_get

The description of i2c\_interrupt\_flag\_get is shown as below:

Table 3-487. Function `i2c_interrupt_flag_get`

<b>Function name</b>	<code>i2c_interrupt_flag_get</code>
<b>Function prototype</b>	<code>FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, uint32_t int_flag);</code>
<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=0,1,2)
<b>Input parameter{in}</b>	
<b>int_flag</b>	interrupt flag
<i>I2C_INT_FLAG_SBSE ND</i>	start condition sent out in master mode interrupt flag
<i>I2C_INT_FLAG_ADDS END</i>	address is sent in master mode or received and matches in slave mode interrupt flag
<i>I2C_INT_FLAG_BTC</i>	byte transmission finishes
<i>I2C_INT_FLAG_ADD10 SEND</i>	header of 10-bit address is sent in master mode interrupt flag
<i>I2C_INT_FLAG_STPD ET</i>	stop condition detected in slave mode interrupt flag
<i>I2C_INT_FLAG_RBNE</i>	I2C_DATA is not Empty during receiving interrupt flag
<i>I2C_INT_FLAG_TBE</i>	I2C_DATA is empty during transmitting interrupt flag
<i>I2C_INT_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
<i>I2C_INT_FLAG_LOSTA RB</i>	arbitration lost in master mode interrupt flag
<i>I2C_INT_FLAG_AERR</i>	acknowledge error interrupt flag
<i>I2C_INT_FLAG_OUER R</i>	over-run or under-run situation occurs in slave mode interrupt flag
<i>I2C_INT_FLAG_PECER RR</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBT</i>	timeout signal in SMBus mode interrupt flag



O	
I2C_INT_FLAG_SMBA LT	SMBus Alert status interrupt flag
I2C_INT_FLAG_TFF	txframe fall interrupt flag
I2C_INT_FLAG_TFR	txframe rise interrupt flag
I2C_INT_FLAG_RFF	rxframe fall interrupt flag
I2C_INT_FLAG_RFR	rxframe rise interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* check the byte transmission finishes interrupt flag is set or not*/
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get (I2C0, I2C_INT_FLAG_BTC);
```

### i2c\_interrupt\_flag\_clear

The description of i2c\_interrupt\_flag\_clear is shown as below:

**Table 3-488. Function i2c\_interrupt\_flag\_clear**

Function name	i2c_interrupt_flag_clear
Function prototype	void i2c_interrupt_flag_clear(uint32_t i2c_periph, uint32_t int_flag);
Function descriptions	clear I2C interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
intflag	interrupt flag
I2C_INT_FLAG_ADDS	address is sent in master mode or received and matches in slave mode

<i>END</i>	interrupt flag
<i>I2C_INT_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
<i>I2C_INT_FLAG_LOSTARB</i>	arbitration lost in master mode interrupt flag
<i>I2C_INT_FLAG_AERR</i>	acknowledge error interrupt flag
<i>I2C_INT_FLAG_OVERR</i>	over-run or under-run situation occurs in slave mode interrupt flag
<i>I2C_INT_FLAG_PECERR</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBTO</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBALTL</i>	SMBus Alert status interrupt flag
<i>I2C_INT_FLAG_TFF</i>	txframe fall interrupt flag
<i>I2C_INT_FLAG_TFR</i>	txframe rise interrupt flag
<i>I2C_INT_FLAG_RFF</i>	rxframe fall interrupt flag
<i>I2C_INT_FLAG_RFR</i>	rxframe rise interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the acknowledge error interrupt flag */
```

```
i2c_interrupt_flag_clear (I2C0, I2C_INT_FLAG_AERR);
```

## 3.17. IPA

The IPA provides a configurable and flexible image format conversion from one or two source image to the destination image. The IPA registers are listed in chapter [3.17.1](#), the IPA firmware functions are introduced in chapter [3.17.2](#).

### 3.17.1. Descriptions of Peripheral registers

IPA registers are listed in the table shown as below:

**Table 3-489. IPA Registers**

Registers	Descriptions
IPA_CTL	IPA control register
IPA_INTF	IPA interrupt flag register
IPA_INTC	IPA interrupt flag clear register
IPA_FMADDR	IPA foreground memory base address register
IPA_FLOFF	IPA foreground line offset register
IPA_BMADDR	IPA background memory base address register
IPA_BLOFF	IPA background line offset register
IPA_FPCTL	IPA foreground pixel control register
IPA_FPV	IPA foreground pixel value register
IPA_BPCTL	IPA background pixel control register
IPA_BPV	IPA background pixel value register
IPA_FLMADDR	IPA foreground LUT memory base address register
IPA_BLMADDR	IPA background LUT memory base address register
IPA_DPCTL	IPA destination pixel control register
IPA_DPV	IPA destination pixel value register
IPA_DMADDR	IPA destination memory base address register
IPA_DLOFF	IPA destination line offset register
IPA_IMS	IPA image size register
IPA_LM	IPA line mark register
IPA_ITCTL	IPA inter-timer control register

### 3.17.2. Descriptions of Peripheral functions

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

**Table 3-490. IPA firmware function**

Function name	Function description
ipa_deinit	deinitialize IPA
ipa_transfer_enable	enable IPA transfer
ipa_transfer_hangup_enable	enable IPA transfer hang up
ipa_transfer_hangup_disable	disable IPA transfer hang up
ipa_transfer_stop_enable	enable IPA transfer stop
ipa_transfer_stop_disable	disable IPA transfer stop
ipa_foreground_lut_loading_enable	enable IPA foreground LUT loading
ipa_background_lut_loading_enable	enable IPA background LUT loading
ipa_pixel_format_convert_mode_set	set pixel format convert mode, the function is invalid when the IPA transfer is enabled
ipa_foreground_struct_para_init	initialize the structure of IPA foreground parameter struct with the default values, it is suggested that call this function after an ipa_foreground_parameter_struct structure is defined
ipa_foreground_init	initialize foreground parameters
ipa_background_struct_para_init	initialize the structure of IPA background parameter struct with the default values, it is suggested that call this function after an ipa_background_parameter_struct structure is defined
ipa_background_init	initialize background parameters
ipa_destination_struct_para_init	initialize the structure of IPA destination parameter struct with the default values, it is suggested that call this function after an ipa_destination_parameter_struct structure is defined
ipa_destination_init	initialize destination parameters
ipa_foreground_lut_init	initialize IPA foreground LUT parameters
ipa_background_lut_init	initialize IPA background LUT parameters
ipa_line_mark_config	configure IPA line mark
ipa_inter_timer_config	IPA inter-timer enable or disable
ipa_interval_clock_num_config	configure the number of clock cycles interval
ipa_flag_get	get IPA flag status in IPA_INTF register
ipa_flag_clear	clear IPA flag in IPA_INTF register

Function name	Function description
ipa_interrupt_enable	enable IPA interrupt
ipa_interrupt_disable	disable IPA interrupt
ipa_interrupt_flag_get	get IPA interrupt flag
ipa_interrupt_flag_clear	clear IPA interrupt flag

### Structure ipa\_foreground\_parameter\_struct

**Table3-491. Structure ipa\_foreground\_parameter\_struct**

Member name	Function description
foreground_memaddr	foreground memory base address
foreground_lineoff	foreground line offset
foreground_prealpha	foreground pre-defined alpha value
foreground_alpha_algorithm	foreground alpha value calculation algorithm
foreground_pf	foreground pixel format
foreground_prered	foreground pre-defined red value
foreground_pregreen	foreground pre-defined green value
foreground_preblue	foreground pre-defined blue value

### Structure ipa\_background\_parameter\_struct

**Table3-492. Structure ipa\_background\_parameter\_struct**

Member name	Function description
background_memaddr	background memory base address
background_lineoff	background line offset
background_prealpha	background pre-defined alpha value
background_alpha_algorithm	background alpha value calculation algorithm
background_pf	background pixel format
background_prered	background pre-defined red value
background_pregreen	background pre-defined green value
background_preblue	background pre-defined blue value

## Structure ipa\_destination\_parameter\_struct

**Table3-493. Structure ipa\_destination\_parameter\_struct**

Member name	Function description
destination_memaddr	destination memory base address
destination_lineoff	destination line offset
destination_prealpha	destination pre-defined alpha value
destination_pf	destination pixel format
destination_prered	destination pre-defined red value
destination_pregreen	destination pre-defined green value
destination_preblue	destination pre-defined blue value
image_width	width of the image to be processed
image_height	height of the image to be processed

## Enum ipa\_dpf\_enum

**Table 3-494. Enum ipa\_dpf\_enum**

enum name	Function description
IPA_DPF_ARGB8888	destination pixel format ARGB8888
IPA_DPF_RGB888	destination pixel format RGB888
IPA_DPF_RGB565	destination pixel format RGB565
IPA_DPF_ARGB1555	destination pixel format ARGB1555
IPA_DPF_ARGB4444	destination pixel format ARGB4444

## ipa\_deinit

The description of ipa\_deinit is shown as below:

**Table 3-495. Function ipa\_deinit**

Function name	ipa_deinit
Function prototype	void ipa_deinit(void);
Function descriptions	deinitialize IPA registers
Precondition	-
The called functions	-

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

Example:

```
/* deinitialize IPA */
```

```
ipa_deinit();
```

### ipa\_transfer\_enable

The description of ipa\_transfer\_enable is shown as below:

**Table 3-496. Function ipa\_transfer\_enable**

Function name	ipa_transfer_enable
Function prototype	void ipa_transfer_enable(void);
Function descriptions	enable IPA transfer
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IPA transfer */
```

```
ipa_transfer_enable();
```

### ipa\_transfer\_hangup\_enable

The description of ipa\_transfer\_hangup\_enable is shown as below:

Table 3-497. Function ipa\_transfer\_hangup\_enable

Function name	ipa_transfer_hangup_enable
Function prototype	void ipa_transfer_hangup_enable(void);
Function descriptions	enable IPA transfer hang up
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IPA transfer hang up */
ipa_transfer_hangup_enable();
```

### ipa\_transfer\_hangup\_disable

The description of ipa\_transfer\_hangup\_disable is shown as below:

Table 3-498. Function ipa\_transfer\_hangup\_disable

Function name	ipa_transfer_hangup_disable
Function prototype	void ipa_transfer_hangup_disable(void);
Function descriptions	disable IPA transfer hang up
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-



Example:

```
/* disable IPA transfer hang up */
ipa_transfer_hangup_disable();
```

### ipa\_transfer\_stop\_enable

The description of ipa\_transfer\_stop\_enable is shown as below:

**Table 3-499. Function ipa\_transfer\_stop\_enable**

Function name	ipa_transfer_stop_enable
Function prototype	void ipa_transfer_stop_enable(void);
Function descriptions	enable IPA transfer stop
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IPA transfer stop */
ipa_transfer_stop_enable();
```

### ipa\_transfer\_stop\_disable

The description of ipa\_transfer\_stop\_disable is shown as below:

**Table 3-500. Function ipa\_transfer\_stop\_disable**

Function name	ipa_transfer_stop_disable
Function prototype	void ipa_transfer_stop_disable(void);
Function descriptions	disable IPA transfer stop
Precondition	-
The called functions	-

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

Example:

```
/* disable IPA transfer stop */
```

```
ipa_transfer_stop_disable();
```

### ipa\_foreground\_lut\_loading\_enable

The description of ipa\_foreground\_lut\_loading\_enable is shown as below:

**Table 3-501. Function ipa\_foreground\_lut\_loading\_enable**

Function name	ipa_foreground_lut_loading_enable
Function prototype	void ipa_foreground_lut_loading_enable(void);
Function descriptions	enable IPA foreground LUT loading
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IPA foreground LUT loading */
```

```
ipa_foreground_lut_loading_enable();
```

### ipa\_background\_lut\_loading\_enable

The description of ipa\_background\_lut\_loading\_enable is shown as below:

Table 3-502. Function ipa\_background\_lut\_loading\_enable

Function name	ipa_background_lut_loading_enable
Function prototype	void ipa_background_lut_loading_enable(void);
Function descriptions	enable IPA background LUT loading
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IPA background LUT loading */
ipa_background_lut_loading_enable();
```

### ipa\_pixel\_format\_convert\_mode\_set

The description of ipa\_pixel\_format\_convert\_mode\_set is shown as below:

Table 3-503. Function ipa\_pixel\_format\_convert\_mode\_set

Function name	ipa_pixel_format_convert_mode_set
Function prototype	void ipa_pixel_format_convert_mode_set(uint32_t pfcmm);
Function descriptions	set pixel format convert mode, the function is invalid when the IPA transfer is enabled
Precondition	-
The called functions	-
Input parameter{in}	
pfcmm	pixel format convert mode
IPA_FGTMODE	foreground memory to destination memory without pixel format convert
IPA_FGTMODE_PF_CONVERT	foreground memory to destination memory with pixel format convert

<i>IPA_FGBGTODE</i>	foreground and background memory to destination memory
<i>IPA_FILL_UP_DE</i>	fill up destination memory with specific color
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure foreground memory to destination memory with pixel format convert */
ipa_pixel_format_convert_mode_set(IPA_FGTODE_PF_CONVERT);
```

### ipa\_foreground\_struct\_para\_init

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

**Table 3-504. Function ipa\_foreground\_struct\_para\_init**

<b>Function name</b>	ipa_foreground_struct_para_init
<b>Function prototype</b>	void ipa_foreground_struct_para_init(ipa_foreground_parameter_struct* foreground_struct);
<b>Function descriptions</b>	initialize the structure of IPA foreground parameter struct with the default values, it is suggested that call this function after an ipa_foreground_parameter_struct structure is defined
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>*foreground_struct</b>	a pointer to ipa_foreground_parameter_struct
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
ipa_foreground_parameter_struct fg_struct;

/* initialize the structure of IPA foreground parameter struct with the default values */
ipa_foreground_struct_para_init(&fg_struct);
```

## ipa\_foreground\_init

The description of ipa\_foreground\_init is shown as below:

**Table 3-505. Function ipa\_foreground\_init**

<b>Function name</b>	ipa_foreground_init
<b>Function prototype</b>	void ipa_foreground_init(ipa_foreground_parameter_struct* foreground_struct);
<b>Function descriptions</b>	initialize foreground parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>*foreground_struct</b>	a pointer to ipa_foreground_parameter_struct
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
ipa_foreground_parameter_struct  ipa_fg_init_struct;

ipa_foreground_struct_para_init(&ipa_fg_init_struct);

/* configure IPA foreground */

ipa_fg_init_struct.foreground_memaddr = (uint32_t)&gBuffer;

ipa_fg_init_struct.foreground_pf = FOREGROUND_PPF_RGB565;

ipa_fg_init_struct.foreground_alpha_algorithm = IPA_FG_ALPHA_MODE_1;

ipa_fg_init_struct.foreground_prealpha = 0x75;

ipa_fg_init_struct.foreground_lineoff = 0x00;

ipa_fg_init_struct.foreground_preblue = 0x00;

ipa_fg_init_struct.foreground_pregreen = 0x00;

ipa_fg_init_struct.foreground_prered = 0x00;

/* foreground initialization */

ipa_foreground_init(&ipa_fg_init_struct);
```

## ipa\_background\_struct\_para\_init

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

**Table 3-506. Function ipa\_background\_struct\_para\_init**

<b>Function name</b>	ipa_background_struct_para_init
<b>Function prototype</b>	void ipa_background_struct_para_init(ipa_background_parameter_struct* background_struct);
<b>Function descriptions</b>	initialize the structure of IPA background parameter struct with the default values , it is suggested that call this function after an ipa_background_parameter_struct structure is defined
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>*background_struct</b>	a pointer to ipa_background_parameter_struct
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
ipa_background_parameter_struct bg_struct;
```

```
/* initialize the structure of IPA background parameter struct with the default values */
```

```
ipa_background_struct_para_init(&bg_struct);
```

## ipa\_background\_init

The description of ipa\_background\_init is shown as below:

**Table 3-507. Function ipa\_background\_init**

<b>Function name</b>	ipa_background_init
<b>Function prototype</b>	void ipa_background_init(ipa_background_parameter_struct* background_struct);
<b>Function descriptions</b>	initialize background parameters
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>*background_struct</b>	a pointer to ipa_background_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

ipa_background_parameter_struct ipa_bg_init_struct;

ipa_background_struct_para_init(&ipa_bg_init_struct);

/* configure IPA background */

ipa_bg_init_struct.background_memaddr = (uint32_t)&gBuffer;

ipa_bg_init_struct.background_pf = BACKGROUND_PPF_RGB565;

ipa_bg_init_struct.background_alpha_algorithm = IPA_BG_ALPHA_MODE_0;

ipa_bg_init_struct.background_prealpha = 255;

ipa_bg_init_struct.background_lineoff = 0x00;

ipa_bg_init_struct.background_preblue = 0x00;

ipa_bg_init_struct.background_pregreen = 0x00;

ipa_bg_init_struct.background_prered = 0x00;

/* background initialization */

ipa_background_init(&ipa_bg_init_struct);

```

### ipa\_destination\_struct\_para\_init

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

**Table 3-508. Function ipa\_destination\_struct\_para\_init**

Function name	ipa_destination_struct_para_init
Function prototype	void ipa_destination_struct_para_init(ipa_destination_parameter_struct* destination_struct);
Function descriptions	initialize the structure of IPA destination parameter struct with the default values, it is suggested that call this function after an ipa_destination_parameter_struct structure is defined
Precondition	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>*destination_struct</b>	a pointer to ipa_destination_parameter_struct
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
ipa_destination_parameter_struct destination_struct;
```

```
/* initialize the structure of IPA destination parameter struct with the default values */
```

```
ipa_destination_struct_para_init(&destination_struct);
```

### ipa\_destination\_init

The description of ipa\_destination\_init is shown as below:

**Table 3-509. Function ipa\_destination\_init**

<b>Function name</b>	ipa_destination_init
<b>Function prototype</b>	void ipa_destination_init(ipa_destination_parameter_struct* destination_struct);
<b>Function descriptions</b>	initialize destination parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>*destination_struct</b>	a pointer to ipa_destination_parameter_struct
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
ipa_destination_parameter_struct ipa_destination_init_struct;
```



```

ipa_destination_struct_para_init(&ipa_destination_init_struct);

/* configure destination pixel format */

ipa_destination_init_struct.destination_pf = IPA_DPF_RGB565;

/* configure destination memory base address */

ipa_destination_init_struct.destination_memaddr = (uint32_t)&gBuffer;

/* configure destination pre-defined alpha value RGB */

ipa_destination_init_struct.destination_pregreen = 0;

ipa_destination_init_struct.destination_preblue = 0;

ipa_destination_init_struct.destination_prered = 0;

ipa_destination_init_struct.destination_prealpha = 0;

/* configure destination line offset */

ipa_destination_init_struct.destination_lineoff = 0;

/* configure height of the image to be processed */

ipa_destination_init_struct.image_height = 160;

/* configure width of the image to be processed */

ipa_destination_init_struct.image_width = 229;

/* ipa destination initialization */

ipa_destination_init(&ipa_destination_init_struct);

```

### ipa\_foreground\_lut\_init

The description of ipa\_foreground\_lut\_init is shown as below:

**Table 3-510. Function ipa\_foreground\_lut\_init**

Function name	ipa_foreground_lut_init
Function prototype	void ipa_foreground_lut_init(uint8_t fg_lut_num, uint8_t fg_lut_pf, uint32_t fg_lut_addr);
Function descriptions	initialize IPA foreground LUT parameters
Precondition	-
The called functions	-
Input parameter{in}	
fg_lut_num	foreground LUT number of pixel

Input parameter{in}	
<b>fg_lut_pf</b>	foreground LUT pixel format
<i>IPA_LUT_PF_ARGB8888</i>	LUT pixel format is ARGB8888
<i>IPA_LUT_PF_RGB888</i>	LUT pixel format is RGB888
Input parameter{in}	
<b>fg_lut_addr</b>	foreground LUT memory base address. The address must be aligned to 8-bit, 16-bit or 32-bit corresponding with the foreground LUT pixel format.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the LUT foreground LUT parameters */
```

```
ipa_foreground_lut_init(1, IPA_LUT_PF_ARGB8888, 0x20002000);
```

### ipa\_background\_lut\_init

The description of ipa\_background\_lut\_init is shown as below:

**Table 3-511. Function ipa\_background\_lut\_init**

<b>Function name</b>	ipa_background_lut_init
<b>Function prototype</b>	void ipa_background_lut_init(uint8_t bg_lut_num, uint8_t bg_lut_pf, uint32_t bg_lut_addr);
<b>Function descriptions</b>	initialize IPA background LUT parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>bg_lut_num</b>	background LUT number of pixel
Input parameter{in}	
<b>bg_lut_pf</b>	background LUT pixel format
<i>IPA_LUT_PF_ARGB8888</i>	LUT pixel format is ARGB8888
<i>88</i>	

<i>IPA_LUT_PF_RGB888</i>	LUT pixel format is RGB888
<b>Input parameter{in}</b>	
<b>bg_lut_addr</b>	background LUT memory base address. The address must be aligned to 8-bit, 16-bit or 32-bit corresponding with the foreground LUT pixel format.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the LUT background LUT parameters */
```

```
ipa_background_lut_init(2, IPA_LUT_PF_RGB888, 0x20001000);
```

### ipa\_line\_mark\_config

The description of ipa\_line\_mark\_config is shown as below:

**Table 3-512. Function ipa\_line\_mark\_config**

<b>Function name</b>	ipa_line_mark_config
<b>Function prototype</b>	void ipa_line_mark_config(uint16_t line_num);
<b>Function descriptions</b>	configure IPA line mark
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>line_num</b>	line number
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure line mark */
```

```
ipa_line_mark_config(10);
```

## ipa\_inter\_timer\_config

The description of ipa\_inter\_timer\_config is shown as below:

**Table 3-513. Function ipa\_inter\_timer\_config**

<b>Function name</b>	ipa_inter_timer_config
<b>Function prototype</b>	void ipa_inter_timer_config(uint8_t timer_cfg);
<b>Function descriptions</b>	inter-timer enable or disable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_cfg</b>	inter-timer configuration
IPA_INTER_TIMER_ENABLE	enable inter-timer
IPA_INTER_TIMER_DISABLE	disable inter-timer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable inter-timer */
```

```
ipa_inter_timer_config(IPA_INTER_TIMER_ENABLE);
```

## ipa\_interval\_clock\_num\_config

The description of ipa\_interval\_clock\_num\_config is shown as below:

**Table 3-514. Function ipa\_interval\_clock\_num\_config**

<b>Function name</b>	ipa_interval_clock_num_config
<b>Function prototype</b>	void ipa_interval_clock_num_config(uint8_t clk_num);
<b>Function descriptions</b>	configure the number of clock cycles interval
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
clk_num	the number of clock cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the number of clock cycles interval and it has no meaning if ITEN is '0' */
```

```
ipa_interval_clock_num_config(1);
```

### ipa\_flag\_get

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

**Table 3-515. Function ipa\_flag\_get**

Function name	ipa_flag_get
Function prototype	FlagStatus ipa_flag_get(uint32_t flag);
Function descriptions	get IPA flag status in IPA_INTF register
Precondition	-
The called functions	-
Input parameter{in}	
flag	IPA flags
IPA_FLAG_TAE	transfer access error interrupt flag
IPA_FLAG_FTF	full transfer finish interrupt flag
IPA_FLAG_TLM	transfer line mark interrupt flag
IPA_FLAG_LAC	LUT access conflict interrupt flag
IPA_FLAG_LLF	LUT loading finish interrupt flag
IPA_FLAG_WCF	wrong configuration interrupt flag
Output parameter{out}	
-	-
Return value	

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

Example:

```
/* wait for full transfer finish flag set */
while(ipa_flag_get(IPA_FLAG_FTF) == RESET);
```

### ipa\_flag\_clear

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

**Table 3-516. Function ipa\_flag\_clear**

<b>Function name</b>	ipa_flag_clear
<b>Function prototype</b>	void ipa_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear IPA flag in IPA_INTF register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	IPA flags
<i>IPA_FLAG_TAE</i>	transfer access error interrupt flag
<i>IPA_FLAG_FTF</i>	full transfer finish interrupt flag
<i>IPA_FLAG_TLM</i>	transfer line mark interrupt flag
<i>IPA_FLAG_LAC</i>	LUT access conflict interrupt flag
<i>IPA_FLAG_LLF</i>	LUT loading finish interrupt flag
<i>IPA_FLAG_WCF</i>	wrong configuration interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* wait for full transfer finish flag set */
while(ipa_flag_get(IPA_FLAG_FTF) == RESET);

/* clear full transfer finish flag */
ipa_flag_clear(IPA_FLAG_FTF);
```

## ipa\_interrupt\_enable

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

**Table 3-517. Function ipa\_interrupt\_enable**

<b>Function name</b>	ipa_interrupt_enable
<b>Function prototype</b>	void ipa_interrupt_enable(uint32_t int_flag);
<b>Function descriptions</b>	enable IPA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	IPA interrupt flags
<i>IPA_INT_TAE</i>	transfer access error interrupt
<i>IPA_INT_FTF</i>	full transfer finish interrupt
<i>IPA_INT_TLM</i>	transfer line mark interrupt
<i>IPA_INT_LAC</i>	LUT access conflict interrupt
<i>IPA_INT_LLF</i>	LUT loading finish interrupt
<i>IPA_INT_WCF</i>	wrong configuration interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
ipa_interrupt_enable(IPA_INT_FTF);
```

## ipa\_interrupt\_disable

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

**Table 3-518. Function ipa\_interrupt\_disable**

<b>Function name</b>	ipa_interrupt_disable
<b>Function prototype</b>	void ipa_interrupt_disable(uint32_t int_flag);
<b>Function descriptions</b>	disable IPA interrupt

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	IPA interrupt flags
<i>IPA_INT_TAE</i>	transfer access error interrupt
<i>IPA_INT_FTF</i>	full transfer finish interrupt
<i>IPA_INT_TLM</i>	transfer line mark interrupt
<i>IPA_INT_LAC</i>	LUT access conflict interrupt
<i>IPA_INT_LLF</i>	LUT loading finish interrupt
<i>IPA_INT_WCF</i>	wrong configuration interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
ipa_interrupt_disable(IPA_INT_FTF);
```

### ipa\_interrupt\_flag\_get

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

**Table 3-519. Function ipa\_interrupt\_flag\_get**

<b>Function name</b>	ipa_interrupt_flag_get
<b>Function prototype</b>	FlagStatus ipa_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	get IPA interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	IPA interrupt flag flags
<i>IPA_INT_FLAG_TAE</i>	transfer access error interrupt flag



<i>IPA_INT_FLAG_FTF</i>	full transfer finish interrupt flag
<i>IPA_INT_FLAG_TLM</i>	transfer line mark interrupt flag
<i>IPA_INT_FLAG_LAC</i>	LUT access conflict interrupt flag
<i>IPA_INT_FLAG_LLF</i>	LUT loading finish interrupt flag
<i>IPA_INT_FLAG_WCF</i>	wrong configuration interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* check whether full transfer finish interrupt flag is SET */
```

```
while(ipa_interrupt_flag_get(IPA_INT_FLAG_FTF) == RESET);
```

### ipa\_interrupt\_flag\_clear

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

**Table 3-520. Function ipa\_interrupt\_flag\_clear**

<b>Function name</b>	ipa_interrupt_flag_clear
<b>Function prototype</b>	void ipa_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear IPA interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	IPA interrupt flag flags
<i>IPA_INT_FLAG_TAE</i>	transfer access error interrupt flag
<i>IPA_INT_FLAG_FTF</i>	full transfer finish interrupt flag
<i>IPA_INT_FLAG_TLM</i>	transfer line mark interrupt flag
<i>IPA_INT_FLAG_LAC</i>	LUT access conflict interrupt flag
<i>IPA_INT_FLAG_LLF</i>	LUT loading finish interrupt flag
<i>IPA_INT_FLAG_WCF</i>	wrong configuration interrupt flag

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
if(ipa_interrupt_flag_get(IPA_INT_FLAG_FTF) != RESET){
    /* clear full transfer finish interrupt flag */
    ipa_interrupt_flag_clear(IPA_INT_FLAG_FTF);
}
```

## 3.18. IREF

IREF is a software package that provide the interfaces for IREF. The IREF registers are listed in chapter [3.18.1](#), the IREF firmware functions are introduced in chapter [3.18.2](#).

### 3.18.1. Descriptions of Peripheral registers

Table 3-521. IREF Registers

Registers	Descriptions
IREF_CTL	Control register

### 3.18.2. Descriptions of Peripheral functions

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

Table 3-522. IREF firmware function

Function name	Function description
iref_deinit	deinit IREF
iref_enable	enable IREF
iref_disable	disable IREF
iref_mode_set	set IREF mode
iref_sink_set	set IREF sink mode
iref_precision_trim_value_set	set IREF precision_trim_value
iref_step_data_config	set IREF step data

#### iref\_deinit

The description of iref\_deinit is shown as below:

Table 3-523. Function iref\_deinit

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

Example:

```
/* reset IREF */
```

```
iref_deinit();
```

### iref\_enable

The description of iref\_enable is shown as below:

Table 3-524. Function iref\_enable

Function name	iref_enable
Function prototype	void iref_enable (void);
Function descriptions	Enable IREF
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IREF */
```

```
iref_enable();
```

## iref\_disable

The description of iref\_disable is shown as below:

**Table 3-525. Function iref\_disable**

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

Example:

```
/* disable IREF */
```

```
iref_disable();
```

## iref\_mode\_set

The description of iref\_mode\_set is shown as below:

**Table 3-526. Function iref\_mode\_set**

<b>Function name</b>	iref_mode_set
<b>Function prototype</b>	void iref_mode_set(uint32_t step);
<b>Function descriptions</b>	set IREF mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>step</b>	IREF mode
<i>IREF_MODE_LOW_POWER</i>	Low power mode,1uA step
<i>IREF_MODE_HIGH_CURRENT</i>	High current mode,8uA step
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* set IREF mode */
iref_mode_set(IREF_MODE_LOW_POWER);
```

### iref\_sink\_set

The description of iref\_sink\_set is shown as below:

**Table 3-527. Function iref\_sink\_set**

<b>Function name</b>	iref_sink_set
<b>Function prototype</b>	void iref_sink_set(uint32_t sinkmode);
<b>Function descriptions</b>	set IREF sink mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sinkmode</b>	Sink mode
<i>IREF_SOURCE_CURRENT</i>	source current
<i>IREF_SINK_CURRENT</i>	sink current
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set IREF source current mode */
iref_sink_set(IREF_SOURCE_CURRENT);
```

### iref\_precision\_trim\_value\_set

The description of iref\_precision\_trim\_value\_set is shown as below:

**Table 3-528. Function iref\_precision\_trim\_value\_set**

<b>Function name</b>	iref_precision_trim_value_set
<b>Function prototype</b>	void iref_precision_trim_value_set (uint32_t precisiontrim);
<b>Function descriptions</b>	set IREF precision trim value
<b>Precondition</b>	-
<b>The called</b>	-

functions	
Input parameter{in}	
<b>precisiontrim</b>	IREF precision_trim_value
<i>IREF_CUR_PRECISION_TRIM_X</i>	(X=0..31): (-15.. +16)%
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set IREF precision trim value -15% */
```

```
iref_precision_trim_value_set(IREF_CUR_PRECISION_TRIM_0);
```

### iref\_step\_data\_config

The description of iref\_step\_data\_config is shown as below:

**Table 3-529. Function iref\_step\_data\_config**

<b>Function name</b>	iref_step_data_config
<b>Function prototype</b>	void iref_step_data_config (uint32_t stepdata);
<b>Function descriptions</b>	set IREF step data
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>stepdata</b>	Step data
<i>IREF_CUR_STEP_DATA_X</i>	(X=0..63): step*x
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set IREF step data */
```

```
iref_step_data_config(IREF_CUR_STEP_DATA_0);
```

## 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 3-530. 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\_cm4.h file

2. refer to the structure SCB\_Type, is defined in the core\_cm4.h file

**Table 3-531. 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 3-532. Enum IRQn\_Type

Member name	Function description
WWDGT_IRQn	window watchDog timer interrupt
LVD_IRQn	LVD through EXTI line detect interrupt
TAMPER_STAMP_IRQn	tamper through EXTI line detect
RTC_WKUP_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
ADC_IRQn	ADC interrupt
CAN0_TX_IRQn	CAN0 transmit interrupts
CAN0_RX0_IRQn	CAN0 receive0 interrupts
CAN0_RX1_IRQn	CAN0 receive1 interrupts
CAN0_EWMC_IRQn	CAN0 EWMC interrupts
EXTI5_9_IRQn	EXTI[9:5] interrupts
TIMER0_BRK_TIMER8_IRQn	TIMER0 break and TIMER8 interrupts
TIMER0_UP_TIMER9_IRQn	TIMER0 update and TIMER9 interrupts
TIMER0_TRG_CMT_TIMER10_IRQn	TIMER0 trigger and commutation and TIMER10 interrupts
TIMER0_Channel_IRQn	TIMER0 channel capture compare interrupt
TIMER1_IRQn	TIMER1 interrupt
TIMER2_IRQn	TIMER2 interrupt
TIMER3_IRQn	TIMER3 interrupt
I2C0_EV_IRQn	I2C0 event interrupt
I2C0_ER_IRQn	I2C0 error interrupt
I2C1_EV_IRQn	I2C1 event interrupt
I2C1_ER_IRQn	I2C1 error interrupt
SPI0_IRQn	SPI0 interrupt



Member name	Function description
SPI1_IRQn	SPI1 interrupt
USART0_IRQn	USART0 interrupt
USART1_IRQn	USART1 interrupt
USART2_IRQn	USART2 interrupt
EXTI10_15_IRQn	EXTI[15:10] interrupts
RTC_Alarm_IRQn	RTC alarm interrupt
USBFS_WKUP_IRQn	USBFS wakeup interrupt
TIMER7_BRK_TIMER11_IRQn	TIMER7 break and TIMER11 interrupts
TIMER7_UP_TIMER12_IRQn	TIMER7 update and TIMER12 interrupts
TIMER7_TRG_CMT_TIMER13_IRQn	TIMER7 trigger and commutation and TIMER13 interrupts
TIMER7_Channel_IRQn	TIMER7 channel capture compare interrupt
DMA0_Channel7_IRQn	DMA0 channel7 interrupt
EXMC_IRQn	EXMC global interrupt
SDIO_IRQn	SDIO global interrupt
TIMER4_IRQn	TIMER4 global interrupt
SPI2_IRQn	SPI2 global interrupt
UART3_IRQn	UART3 global interrupt
UART4_IRQn	UART4 global interrupt
TIMER5_DAC_IRQn	TIMER5 and DAC global interrupt
TIMER6_IRQn	TIMER6 global interrupt
DMA1_Channel0_IRQn	DMA1 channel0 global interrupt
DMA1_Channel1_IRQn	DMA1 channel1 global interrupt
DMA1_Channel2_IRQn	DMA1 channel2 global interrupt
DMA1_Channel3_IRQn	DMA1 channel3 global interrupt
DMA1_Channel4_IRQn	DMA1 channel4 global interrupt
ENET_IRQn	ENET interrupt
ENET_WKUP_IRQn	ENET wakeup interrupt
CAN1_TX_IRQn	CAN1 transmit interrupt
CAN1_RX0_IRQn	CAN1 receive0 interrupt
CAN1_RX1_IRQn	CAN1 receive1 interrupt
CAN1_EWMC_IRQn	CAN1 EWMC interrupt
USBFS_IRQn	USBFS interrupt
DMA1_Channel5_IRQn	DMA1 Channel5 interrupt
DMA1_Channel6_IRQn	DMA1 Channel6 interrupt
DMA1_Channel7_IRQn	DMA1 Channel7 interrupt
USART5_IRQn	USART5 interrupt
I2C2_EV_IRQn	I2C2 event interrupt
I2C2_ER_IRQn	I2C3 error interrupt
USBHS_EP1_Out_IRQn	USBHS endpoint 1 out interrupt
USBHS_EP1_In_IRQn	USBHS endpoint 1 in interrupt
USBHS_WKUP_IRQn	USBHS Wakeup interrupt
USBHS_IRQn	USBHS interrupt

Member name	Function description
DCI_IRQn	DCI interrupt
TRNG_IRQn	TRNG interrupt
FPU_IRQn	FPU interrupt
UART6_IRQn	UART6 interrupt
UART7_IRQn	UART7 interrupt
SPI3_IRQn	SPI3 interrupt
SPI4_IRQn	SPI4 interrupt
SPI5_IRQn	SPI5 interrupt
TLI_IRQn	TLI interrupt
TLI_ER_IRQn	TLI error interrupt
IPA_IRQn	IPA interrupt

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

**Table 3-533. MISC firmware function**

Function name	Function description
nvic_priority_group_set	set the priority group
nvic_irq_enable	enable NVIC interrupt request
nvic_irq_disable	disable NVIC interrupt request
nvic_vector_table_set	set the NVIC vector table address
system_lowpower_set	set the state of the low power mode
system_lowpower_reset	reset the state of the low power mode
systick_clksource_set	set the systick clock source

## nvic\_priority\_group\_set

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

**Table 3-534. Function nvic\_priority\_group\_set**

Function name	nvic_priority_group_set
Function prototype	void nvic_priority_group_set(uint32_t nvic_prigroup);
Function descriptions	configure bits length of the priority group
Precondition	-
The called functions	-
Input parameter{in}	
nvic_prigroup	priority group
NVIC_PRIGROUP_PR E0_SUB4	0 bits for pre-emption priority 4 bits for subpriority
NVIC_PRIGROUP_PR E1_SUB3	1 bits for pre-emption priority 3 bits for subpriority
NVIC_PRIGROUP_PR E2_SUB2	2 bits for pre-emption priority 2 bits for subpriority
NVIC_PRIGROUP_PR E3_SUB1	3 bits for pre-emption priority 1 bits for subpriority

<i>NVIC_PRIGROUP_PR</i> <i>E4_SUB0</i>	4 bits for pre-emption priority 0 bits for subpriority
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* priority group configuration , 0 bits for pre-emption priority 4 bits for subpriority */
```

```
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

### nvic\_irq\_enable

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

**Table 3-535. Function nvic\_irq\_enable**

<b>Function name</b>	nvic_irq_enable
<b>Function prototype</b>	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
<b>Function descriptions</b>	enable NVIC request, configure the priority of interrupt
<b>Precondition</b>	-
<b>The called functions</b>	nvic_priority_group_set
<b>Input parameter{in}</b>	
nvic_irq	NVIC interrupt, refer to enum <a href="#">Enum IRQn_Type</a>
<b>Input parameter{in}</b>	
nvic_irq_pre_priority	the pre-emption priority needed to set (0~4)
<b>Input parameter{in}</b>	
nvic_irq_sub_priority	the subpriority needed to set (0~4)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable window watchDog timer interrupt , pre-emption priority is 1, subpriority is 1 */
```

```
nvic_irq_enable(WWDGT_IRQn, 1, 1);
```

### nvic\_irq\_disable

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

**Table 3-536. Function nvic\_irq\_disable**

<b>Function name</b>	nvic_irq_disable
<b>Function prototype</b>	void nvic_irq_disable(uint8_t nvic_irq);
<b>Function descriptions</b>	disable NVIC request

Precondition	-
The called functions	-
Input parameter{in}	
nvic_irq	NVIC interrupt, refer to enum <a href="#">Enum IRQn_Type</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable window watchDog timer interrupt */
nvic_irq_disable(WWDGT_IRQn);
```

### nvic\_vector\_table\_set

The description of nvic\_vector\_table\_set is shown as below:

**Table 3-537. Function nvic\_vector\_table\_set**

Function name	nvic_vector_table_set
Function prototype	void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);
Function descriptions	set the NVIC vector table address
Precondition	-
The called functions	-
Input parameter{in}	
nvic_vect_tab	the RAM or FLASH base address
NVIC_VECTTAB_RAM	RAM base address
NVIC_VECTTAB_FLASH	Flash base address
Input parameter{in}	
offset	Vector Table offset (vector table start address= base address+offset)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set vector table address = NVIC_VECTTAB_FLASH +0x200 */
nvic_vector_table_set (NVIC_VECTTAB_FLASH, 0x200);
```

### system\_lowpower\_set

The description of system\_lowpower\_set is shown as below:

**Table 3-538. Function system\_lowpower\_set**

Function name	system_lowpower_set
---------------	---------------------

<b>Function prototype</b>	void system_lowpower_set(uint8_t lowpower_mode);
<b>Function descriptions</b>	the state of the low power mode management
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowpower_mode</b>	the low power mode state
SCB_LPM_SLEEP_EXIT_ISR	if chose this para, the system always enter low power mode by exiting from ISR
SCB_LPM_DEEPSLEEP_P	if chose this para, the system will enter the DEEPSLEEP mode
SCB_LPM_WAKE_BY_ALL_INT	if chose this para, the lowpower mode can be woke up by all the enable and disable interrupts
<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 3-539. Function system\_lowpower\_reset**

<b>Function name</b>	system_lowpower_reset
<b>Function prototype</b>	void system_lowpower_reset(uint8_t lowpower_mode);
<b>Function descriptions</b>	the state of the low power mode management
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowpower_mode</b>	the low power mode state
SCB_LPM_SLEEP_EXIT_ISR	if chose this para, the system will exit low power mode by exiting from ISR
SCB_LPM_DEEPSLEEP_P	if chose this para, the system will enter the SLEEP mode
SCB_LPM_WAKE_BY_ALL_INT	if chose this para, the lowpower mode only can be 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 3-540. Function systick\_clksource\_set**

<b>Function name</b>	systick_clksource_set
<b>Function prototype</b>	void systick_clksource_set(uint32_t systick_clksource);
<b>Function descriptions</b>	set the systick clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>systick_clksource</b>	the systick clock source needed to choose
SYSTICK_CLKSOURC E_HCLK	systick clock source is from HCLK
SYSTICK_CLKSOURC E_HCLK_DIV8	systick clock source is from HCLK/8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* systick clock source is HCLK/8 */
systick_clksource_set (SYSTICK_CLKSOURCE_HCLK_DIV8);
```

## 3.20. 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 3-541. PMU Registers**

Registers	Descriptions
PMU_CTL	Control register
PMU_CS	Control and status register

### 3.20.2. Descriptions of Peripheral functions

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

**Table 3-542. PMU firmware function**

Function name	Function description
pmu_deinit	reset PMU registers
pmu_lvd_select	select low voltage detector threshold
pmu_lvd_disable	disable PMU lvd
pmu_ldo_output_select	select LDO output voltage
pmu_highdriver_mode_enable	enable high-driver mode
pmu_highdriver_mode_disable	disable high-driver mode
pmu_highdriver_switch_select	switch high-driver mode
pmu_lowdriver_mode_enable	enable low-driver mode in deep-sleep
pmu_lowdriver_mode_disable	disable low-driver mode in deep-sleep
pmu_lowpower_driver_config	in deep-sleep mode, driver mode when use low power LDO
pmu_normalpower_driver_config	in deep-sleep mode, driver mode when use normal power LDO
pmu_to_sleepmode	PMU work in sleep mode
pmu_to_deepsleepmode	PMU work in deepsleep mode
pmu_to_standbymode	pmu work in standby mode
pmu_wakeup_pin_enable	enable PMU wakeup pin
pmu_wakeup_pin_disable	disable PMU wakeup pin
pmu_backup_ldo_config	backup SRAM LDO on
pmu_backup_write_enable	enable backup domain write
pmu_backup_write_disable	disable backup domain write
pmu_flag_get	get flag status
pmu_flag_clear	clear flag bit

#### pmu\_deinit

The description of pmu\_deinit is shown as below:

**Table 3-543. Function pmu\_deinit**

Function name	pmu_deinit
Function prototype	void pmu_deinit(void);
Function descriptions	reset PMU registers
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset PMU */
```

```
pmu_deinit();
```

### pmu\_lvd\_select

The description of pmu\_lvd\_select is shown as below:

**Table 3-544. Function pmu\_lvd\_select**

Function name	pmu_lvd_select
Function prototype	void pmu_lvd_select(uint32_t lvd_t_n);
Function descriptions	select low voltage detector threshold
Precondition	-
The called functions	-
Input parameter{in}	
lvd_t_n	voltage threshold value
PMU_LVDT_0	voltage threshold is 2.1V
PMU_LVDT_1	voltage threshold is 2.3V
PMU_LVDT_2	voltage threshold is 2.4V
PMU_LVDT_3	voltage threshold is 2.6V
PMU_LVDT_4	voltage threshold is 2.7V
PMU_LVDT_5	voltage threshold is 2.9V
PMU_LVDT_6	voltage threshold is 3.0V
PMU_LVDT_7	voltage threshold is 3.1V
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select low voltage detector threshold as 3.1V */
```

```
pmu_lvd_select (PMU_LVDT_7);
```

### pmu\_lvd\_disable

The description of pmu\_lvd\_disable is shown as below:

**Table 3-545. Function pmu\_lvd\_disable**

Function name	pmu_lvd_disable
Function prototype	void pmu_lvd_disable(void);
Function descriptions	disable PMU LVD
Precondition	-
The called functions	-
Input parameter{in}	



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

Example:

```
/* disable PMU LVD */
```

```
pmu_lvd_disable();
```

### pmu\_ldo\_output\_select

The description of pmu\_ldo\_output\_select is shown as below:

**Table 3-546. 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>	select LDO output voltage
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>ldo_output</b>	LDO output voltage
<i>PMU_LDOVS_LOW</i>	low-driver mode enable in deep-sleep mode
<i>PMU_LDOVS_MID</i>	mid-driver mode disable in deep-sleep mode
<i>PMU_LDOVS_HIGH</i>	high-driver mode disable in deep-sleep mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select low-driver mode as LDO output voltage */
```

```
pmu_ldo_output_select (PMU_LDOVS_LOW);
```

### pmu\_highdriver\_mode\_enable

The description of pmu\_highdriver\_mode\_enable is shown as below:

**Table 3-547. Function pmu\_highdriver\_mode\_enable**

<b>Function name</b>	pmu_highdriver_mode_enable
<b>Function prototype</b>	void pmu_highdriver_mode_enable(void);
<b>Function descriptions</b>	enable high-driver mode
<b>Precondition</b>	-
<b>The called functions</b>	-

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

Example:

```
/* enable high-driver mode */
pmu_highdriver_mode_enable ( );
```

### pmu\_highdriver\_mode\_disable

The description of pmu\_highdriver\_mode\_disable is shown as below:

**Table 3-548. Function pmu\_highdriver\_mode\_disable**

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

Example:

```
/* disable high-driver mode */
pmu_highdriver_mode_disable ( );
```

### pmu\_highdriver\_switch\_select

The description of pmu\_highdriver\_switch\_select is shown as below:

**Table 3-549. Function pmu\_highdriver\_switch\_select**

Function name	pmu_highdriver_switch_select
Function prototype	void pmu_highdriver_switch_select(uint32_t highdr_switch);
Function descriptions	switch high-driver mode
Precondition	-
The called functions	-
Input parameter{in}	
highdr_switch	enable or disable high-driver mode switch

<i>PMU_HIGHDR_SWITC</i> <i>H_NONE</i>	disable high-driver mode switch
<i>PMU_HIGHDR_SWITC</i> <i>H_EN</i>	enable high-driver mode switch
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable high-driver mode switch */
```

```
pmu_highdriver_switch_select (PMU_HIGHDR_SWITCH_EN);
```

### pmu\_lowdriver\_mode\_enable

The description of pmu\_lowdriver\_mode\_enable is shown as below:

**Table 3-550. 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
<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 */
```

```
pmu_lowdriver_mode_enable ();
```

### pmu\_lowdriver\_mode\_disable

The description of pmu\_lowdriver\_mode\_disable is shown as below:

**Table 3-551. 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
<b>Precondition</b>	-
<b>The called functions</b>	-

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

Example:

```
/* disable low-driver mode in deep-sleep */
```

```
pmu_lowdriver_mode_disable ();
```

### pmu\_lowpower\_driver\_config

The description of pmu\_lowpower\_driver\_config is shown as below:

**Table 3-552. Function pmu\_lowpower\_driver\_config**

Function name	pmu_lowpower_driver_config
Function prototype	void pmu_lowpower_driver_config(uint32_t mode);
Function descriptions	in deep-sleep mode, driver mode when use low power LDO
Precondition	-
The called functions	-
Input parameter{in}	
<b>mode</b>	Low power mode of LDO-
<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 1 and use low power LDO
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* normal driver when use low power LDO */
```

```
pmu_lowdriver_lowpower_config (PMU_NORMALDR_LOWPWR);
```

### pmu\_normalpower\_driver\_config

The description of pmu\_normalpower\_driver\_config is shown as below:

**Table 3-553. pmu\_normalpower\_driver\_config**

Function name	pmu_normalpower_driver_config
Function prototype	void pmu_normalpower_driver_config(uint32_t mode);
Function descriptions	in deep-sleep mode, 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 when use normal power LDO-
<i>PMU_NORMALDR_NO RMALPWR</i>	normal driver when use normal power LDO
<i>PMU_LOWDR_NORM ALPWR</i>	low-driver mode enabled when LDEN is 1 and use normal power LDO
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* normal driver when use normal power LDO */
```

```
pmu_lowdriver_normalpower_config (PMU_NORMALDR_NORMALPWR);
```

### pmu\_to\_sleepmode

The description of pmu\_to\_sleepmode is shown as below:

**Table 3-554. Function pmu\_to\_sleepmode**

<b>Function name</b>	pmu_to_sleepmode
<b>Function prototype</b>	void pmu_to_sleepmode(uint8_t sleepmodecmd);
<b>Function descriptions</b>	PMU work in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sleepmodecmd</b>	command to enter sleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* use WFI command to enter sleep mode */
```

```
pmu_to_sleepmode (WFI_CMD);
```

### pmu\_to\_deepsleepmode

The description of pmu\_to\_deepsleepmode is shown as below:

Table 3-555. Function pmu\_to\_deepsleepmode

Function name	pmu_to_deepsleepmode
Function prototype	void pmu_to_deepsleepmode(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmodecmd);
Function descriptions	PMU work in deepsleep mode
Precondition	-
The called functions	-
Input parameter{in}	
ldo	ldo work mode
PMU_LDO_NORMAL	LDO normal work when pmu enter deepsleep mode
PMU_LDO_LOWPOWER	LDO work at low power mode when pmu enter deepsleep mode
Input parameter{in}	
lowdrive	Low-driver mode
PMU_LOWDRIVER_DISABLE	Low-driver mode disable in deep-sleep mode
PMU_LOWDRIVER_ENABLE	Low-driver mode enable in deep-sleep mode
Input parameter{in}	
deepsleepmodecmd	command to enter deepsleep mode
WFI_CMD	use WFI command
WFE_CMD	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* PMU work at deepsleep mode */
```

```
pmu_to_deepsleepmode (PMU_LDO_NORMAL, PMU_LOWDRIVER_ENABLE, WFI_CMD);
```

### pmu\_to\_standbymode

The description of pmu\_to\_standbymode is shown as below:

Table 3-556. Function pmu\_to\_standbymode

Function name	pmu_to_standbymode
Function prototype	void pmu_to_standbymode(void);
Function descriptions	pmu work in standby mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* PMU work at standby mode */
```

```
pmu_to_standbymode ();
```

### pmu\_wakeup\_pin\_enable

The description of pmu\_wakeup\_pin\_enable is shown as below:

**Table 3-557. 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 3-558. 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>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* disable wakeup pin */
pmu_wakeup_pin_disable ();
```

### pmu\_backup\_ldo\_config

The description of pmu\_backup\_ldo\_config is shown as below:

**Table 3-559. Function pmu\_backup\_ldo\_config**

<b>Function name</b>	pmu_backup_ldo_config
<b>Function prototype</b>	void pmu_backup_ldo_config(uint32_t bkp_ldo);
<b>Function descriptions</b>	backup SRAM LDO on
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>bkp_ldo</b>	open or close the backup SRAM LDO
<i>PMU_BLDOON_OFF</i>	backup SRAM LDO closed
<i>PMU_BLDOON_ON</i>	open the backup SRAM LDO
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* backup SRAM LDO on */
pmu_backup_ldo_config (PMU_BLDOON_ON);
```

### pmu\_backup\_write\_enable

The description of pmu\_backup\_write\_enable is shown as below:

**Table 3-560. Function pmu\_backup\_write\_enable**

<b>Function name</b>	pmu_backup_write_enable
<b>Function prototype</b>	void pmu_backup_write_enable (void);
<b>Function descriptions</b>	enable backup domain write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	



-	-
---	---

Example:

```
/* enable backup domain write */
```

```
pmu_backup_write_enable ();
```

### pmu\_backup\_write\_disable

The description of pmu\_backup\_write\_disable is shown as below:

**Table 3-561. Function pmu\_backup\_write\_disable**

<b>Function name</b>	pmu_backup_write_disable
<b>Function prototype</b>	void pmu_backup_write_disable (void);
<b>Function descriptions</b>	disable backup domain write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable backup domain write */
```

```
pmu_backup_write_disable ();
```

### pmu\_flag\_get

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

**Table 3-562. 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>	lvd flag
<i>PMU_FLAG_BLDORF</i>	backup SRAM LDO ready flag
<i>PMU_FLAG_LDOVSRF</i>	LDO voltage select ready flag

<i>PMU_FLAG_HDRF</i>	high-driver ready flag
<i>PMU_FLAG_HDSRF</i>	high-driver switch ready flag
<i>PMU_FLAG_LDRF</i>	low-driver mode ready flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get wakeup flag state*/
pmu_flag_get (PMU_FLAG_WAKEUP);
```

### pmu\_flag\_clear

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

**Table 3-563. Function pmu\_flag\_clear**

<b>Function name</b>	pmu_flag_clear
<b>Function prototype</b>	void pmu_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear flag bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flag
<i>PMU_FLAG_RESET_WAKEUP</i>	reset wakeup flag
<i>PMU_FLAG_RESET_S TANDBY</i>	reset standby flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* clear wakeup flag */
pmu_flag_reset (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 3-564. RCU Registers**

Registers	Descriptions
RCU_CTL	Control register
RCU_PLL	PLL register
RCU_CFG0	Clock configuration register 0
RCU_INT	Clock interrupt register
RCU_AHB1RST	AHB1 reset register
RCU_AHB2RST	AHB2 reset register
RCU_AHB3RST	AHB3 reset register
RCU_APB1RST	APB1 reset register
RCU_APB2RST	APB2 reset register
RCU_AHB1EN	AHB1 enable register
RCU_AHB2EN	AHB2 enable register
RCU_AHB3EN	AHB3 enable register
RCU_APB1EN	APB1 enable register
RCU_APB2EN	APB2 enable register
RCU_AHB1SPEN	AHB1 sleep mode enable register
RCU_AHB2SPEN	AHB2 sleep mode enable register
RCU_AHB3SPEN	AHB3 sleep mode enable register
RCU_APB1SPEN	APB1 sleep mode enable register
RCU_APB2SPEN	APB2 sleep mode enable register
RCU_BDCTL	Backup domain control register
RCU_RSTSCK	Reset source/clock register
RCU_PLLSSCTL	PLL clock spread spectrum control register
RCU_PLLI2S	PLLI2S register
RCU_PLLSAI	PLLSAI register
RCU_CFG1	Configuration register 1

Registers	Descriptions
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_ADDAPB1SPEN	APB1 additional sleep mode enable register
RCU_VKEY	voltage key register
RCU_DSV	Deep-sleep mode voltage register

### 3.21.2. Descriptions of Peripheral functions

**Table 3-565. RCU firmware function**

Function name	Function description
rcu_deinit	deinitialize the RCU
rcu_periph_clock_enable	enable the peripherals clock
rcu_periph_clock_disable	disable the peripherals clock
rcu_periph_clock_sleep_enable	enable the peripherals clock when in sleep mode
rcu_periph_clock_sleep_disable	disable the peripherals clock when in sleep mode
rcu_periph_reset_enable	enable the peripherals reset
rcu_periph_reset_disable	disable the peripheral reset
rcu_bkp_reset_enable	enable the BKP domain reset
rcu_bkp_reset_disable	disable the BKP domain reset
rcu_system_clock_source_config	configure the system clock source
rcu_system_clock_source_get	get the system clock source
rcu_ahb_clock_config	configure the AHB clock prescaler selection
rcu_apb1_clock_config	configure the APB1 clock prescaler selection
rcu_apb2_clock_config	configure the APB2 clock prescaler selection
rcu_ckout0_config	configure the CK_OUT0 clock source
rcu_ckout1_config	configure the CK_OUT1 clock source
rcu_pll_config	configure the main PLL clock

Function name	Function description
rcu_plli2s_config	configure the PLLI2S clock
rcu_pllsai_config	configure the PLLSAI clock
rcu_rtc_clock_config	configure the RTC clock source selection
rcu_rtc_div_config	configure the frequency division of RTC clock when HXTAL was selected as its clock source
rcu_i2s_clock_config	configure the I2S clock source selection
rcu_ck48m_clock_config	configure the CK48M clock selection
rcu_pll48m_clock_config	configure the PLL48M clock selection
rcu_timer_clock_prescaler_config	configure the TIMER clock prescaler selection
rcu_tli_clock_div_config	configure the TLI clock division selection
rcu_flag_get	get the clock stabilization and peripheral reset flags
rcu_all_reset_flag_clear	clear all the reset flag
rcu_interrupt_flag_get	get the clock stabilization interrupt and ckm flags
rcu_interrupt_flag_clear	clear the interrupt flags
rcu_interrupt_enable	enable the stabilization interrupt
rcu_interrupt_disable	disable the stabilization interrupt
rcu_lxtal_drive_capability_config	configure the LXTAL drive capability
rcu_osci_stab_wait	wait for oscillator stabilization flags is SET or oscillator startup is timeout
rcu_osci_on	turn on the oscillator
rcu_osci_off	turn off the oscillator
rcu_osci_bypass_mode_enable	enable the oscillator bypass mode
rcu_osci_bypass_mode_disable	disable the oscillator bypass mode
rcu_hxtal_clock_monitor_enable	enable the HXTAL clock monitor
rcu_hxtal_clock_monitor_disable	disable the HXTAL clock monitor
rcu_irc16m_adjust_value_set	set the IRC8M adjust value
rcu_spread_spectrum_config	configure the spread spectrum modulation for the main PLL clock
rcu_spread_spectrum_enable	enable the spread spectrum modulation for the main PLL

Function name	Function description
	clock
rcu_spread_spectrum_disable	disable the spread spectrum modulation for the main PLL clock
rcu_deepsleep_voltage_set	set the deep-sleep mode voltage value
rcu_voltage_key_unlock	unlock the voltage key
rcu_clock_freq_get	get the system clock, bus clock frequency

### Enum rcu\_periph\_enum

**Table 3-566. Enum rcu\_periph\_enum**

enum name	Function description
RCU_GPIOx	GPIO ports clock (x=A,B,C,D,E,F,G ,H,I)
RCU_CRC	CRC clock
RCU_BKPSRAM	BKPSRAM clock
RCU_TCMRAM	TCMRAM clock
RCU_DMAx	DMAx clock (x=0,1)
RCU_IPA	IPA clock
RCU_ENET	ENET clock
RCU_ENETTX	ENETTX clock
RCU_ENETRX	ENETRX clock
RCU_ENETPTP	ENETPTP clock
RCU_USBHS	USBHS clock
RCU_USBHSULPI	USBHSULPI clock
RCU_DCI	DCI clock
RCU_TRNG	TRNG clock
RCU_USBFS	USBFS clock
RCU_EXMC	EXMC clock
RCU_TIMERx	TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
RCU_WWDGT	WWDGT clock
RCU_SPIx	SPIx clock (x=0,1,2,3,4,5)
RCU_USARTx	USARTx clock (x=0,1,2,5)
RCU_UARTx	UARTx clock (x=3,4 ,6,7)
RCU_I2Cx	I2Cx clock (x=0,1,2)
RCU_CANx	CANx clock (x=0,1)
RCU_PMU	PMU clock
RCU_DAC	DAC clock
RCU_RTC	RTC clock
RCU_ADCx	ADCx clock (x=0,1,2)
RCU_SDIO	SDIO clock
RCU_SYSCFG	SYSCFG interface clock

enum name	Function description
RCU_TLI	TLI clock
RCU_CTC	CTC clock
RCU_IREF	IREF clock

### Enum rcu\_periph\_sleep\_enum

**Table 3-567. Enum rcu\_periph\_sleep\_enum**

enum name	Function description
RCU_GPIOx_SLP	GPIO clock (x=A,B,C,D,E,F,G ,H,I)
RCU_CRC_SLP	CRC clock
RCU_FMC_SLP	FMC clock
RCU_SRAM0_SLP	SRAM0 clock
RCU_SRAM1_SLP	SRAM1 clock
RCU_BKPSRAM	BKPSRAM clock
RCU_SRAM2_SLP	SRAM2 clock
RCU_DMAx_SLP	DMAx clock (x=0,1)
RCU_IPA_SLP	IPA clock
RCU_ENET_SLP	ENET clock
RCU_ENETTX_SLP	ENETTX clock
RCU_ENETRX_SLP	ENETRX clock
RCU_ENETPTP_SLP	ENETPTP clock
RCU_USBHS_SLP	USBHS clock
RCU_USBHSULPI_SLP	USBHSULPI clock
RCU_DCI_SLP	DCI clock
RCU_TRNG_SLP	TRNG clock
RCU_USBFS_SLP	USBFS clock
RCU_EXMC_SLP	EXMC clock
RCU_TIMERx_SLP	TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
RCU_WWDGT_SLP	WWDGT clock
RCU_SPIx_SLP	SPIx clock (x=0,1,2,3,4,5)
RCU_USARTx_SLP	USARTx clock (x=0,1,2,5)
RCU_UARTx_SLP	UARTx clock (x=3,4 ,6,7)
RCU_I2Cx_SLP	I2Cx clock (x=0,1,2)
RCU_CANx_SLP	CANx clock (x=0,1)
RCU_PMU_SLP	PMU clock
RCU_DAC_SLP	DAC clock
RCU_RTC_SLP	RTC clock
RCU_ADCx_SLP	ADCx clock (x=0,1,2)
RCU_SDIO_SLP	SDIO clock
RCU_SYSCFG_SLP	SYSCFG interfaceclock
RCU_TLI_SLP	TLI clock

enum name	Function description
RCU_CTC_SLP	CTC clock
RCU_IREF_SLP	IREF clock

### Enum rcu\_periph\_reset\_enum

**Table 3-568. Enum rcu\_periph\_reset\_enum**

enum name	Function description
RCU_GPIOxRST	reset GPIOx clock (x=A,B,C,D,E,F,G,H,I)
RCU_CRCRST	reset CRC clock
RCU_DMAxRST	reset DMAx clock (x=0,1)
RCU_IPARST	reset IPA clock
RCU_ENETRST	reset ENET clock
RCU_USBHSRST	reset USBHSclock
RCU_DCIRST	reset DCI clock
RCU_TRNGRST	reset TRNG clock
RCU_USBFSRST	reset USBFS clock
RCU_EXMCRST	reset EXMC clock
RCU_TIMERxRST	reset TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
RCU_WWDGTRST	reset WWDGT clock
RCU_SPIxRST	reset SPIx clock (x=0,1,2,3,4,5)
RCU_USARTxRST	reset USARTx clock (x=0,1,2,5)
RCU_UARTxRST	reset UARTx clock (x=3,4,6,7)
RCU_I2CxRST	reset I2Cx clock (x=0,1,2)
RCU_CANxRST	reset CANx clock (x=0,1)
RCU_PMURST	reset PMU clock
RCU_DACRST	reset DAC clock
RCU_ADCxRST	reset ADCx clock (x=0,1,2)
RCU_SDIORST	reset SDIO clock
RCU_SYSCFGRST	reset SYSCFG clock
RCU_TLIRST	reset TLI clock
RCU_CTCRST	reset CAU clock
RCU_IREFRST	reset IREF clock

### Enum rcu\_flag\_enum

**Table 3-569. Enum rcu\_flag\_enum**

enum name	Function description
RCU_FLAG_IRC16MS TB	IRC16M stabilization flag
RCU_FLAG_HXTALST B	HXTAL stabilization flag
RCU_FLAG_PLLI2SSST B	PLLI2S stabilization flag



enum name	Function description
RCU_FLAG_PLLSAIST B	PLLSAI stabilization flag
RCU_FLAG_PLLSTB	PLL stabilization flag
RCU_FLAG_LXTALST B	LXTAL stabilization flag
RCU_FLAG_IRC32KST B	IRC32K stabilization flag
RCU_FLAG_IRC48MS TB	IRC48M stabilization flag
RCU_FLAG_EPRST	external PIN reset flag
RCU_FLAG_PORRST	power reset flag
RCU_FLAG_BORRST	BOR reset flag
RCU_FLAG_SWRST	software reset flag
RCU_FLAG_FWDGTR ST	free watchdog timer reset flag
RCU_FLAG_WWDGTR ST	window watchdog timer reset flag
RCU_FLAG_LPRST	low-power reset flag

### Enum rcu\_int\_flag\_enum

**Table 3-570. Enum rcu\_int\_flag\_enum**

enum name	Function description
RCU_INT_FLAG_IRC3 2KSTB	IRC32K stabilization interrupt flag
RCU_INT_FLAG_LXTA LSTB	LXTAL stabilization interrupt flag
RCU_INT_FLAG_IRC1 6MSTB	IRC16M stabilization interrupt flag
RCU_INT_FLAG_HXTA LSTB	HXTAL stabilization interrupt flag
RCU_INT_FLAG_PLLS TB	PLL stabilization interrupt flag
RCU_INT_FLAG_PLLI2 SSTB	PLLI2S stabilization interrupt flag
RCU_INT_FLAG_PLLS AISTB	PLLSAI stabilization interrupt flag
RCU_INT_FLAG_CKM	HXTAL clock stuck interrupt flag
RCU_INT_FLAG_IRC4 8MSTB	IRC48M stabilization interrupt flag

## Enum rcu\_int\_flag\_clear\_enum

Table 3-571. Enum rcu\_int\_flag\_clear\_enum

enum name	Function description
RCU_INT_FLAG_IRC3 2KSTB_CLR	IRC32K stabilization interrupt flag clear
RCU_INT_FLAG_LXTA LSTB_CLR	LXTAL stabilization interrupt flag clear
RCU_INT_FLAG_IRC1 6MSTB_CLR	IRC16M stabilization interrupt flag clear
RCU_INT_FLAG_HXTA LSTB_CLR	HXTAL stabilization interrupt flag clear
RCU_INT_FLAG_PLLS TB_CLR	PLL stabilization interrupt flag clear
RCU_INT_FLAG_PLLI2 SSTB_CLR	PLLI2S stabilization interrupt flag clear
RCU_INT_FLAG_PLLS AISTB_CLR	PLLSAI stabilization interrupt flag clear
RCU_INT_FLAG_CKM _CLR	clock stuck interrupt flag clear
RCU_INT_FLAG_IRC4 8MSTB_CLR	IRC48M stabilization interrupt flag clear

## Enum rcu\_int\_enum

Table 3-572. Enum rcu\_int\_enum

enum name	Function description
RCU_INT_IRC32KSTB	IRC32K stabilization interrupt enable
RCU_INT_LXTALSTB	LXTAL stabilization interrupt enable
RCU_INT_IRC16MSTB	IRC16M stabilization interrupt enable
RCU_INT_IRC48MSTB	IRC48M stabilization interrupt enable
RCU_INT_HXTALSTB	HXTAL stabilization interrupt enable
RCU_INT_PLLSTB	PLL stabilization interrupt enable
RCU_INT_PLLI2SSTB	PLLI2S stabilization interrupt enable
RCU_INT_PLLSAISTB	PLLSAI stabilization interrupt enable

## Enum rcu\_osci\_type\_enum

Table 3-573. Enum rcu\_osci\_type\_enum

enum name	Function description
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)
RCU_IRC16M	internal 16M RC oscillators(IRC16M)
RCU_IRC48M	internal 48M RC oscillators(IRC48M)

enum name	Function description
RCU_IRC32K	internal 32K RC oscillator(IRC32K)
RCU_PLL_CK	phase locked loop(PLL)
RCU_PLLI2S_CK	PLLI2S phase locked loop
RCU_PLLSAI_CK	PLLSAI phase locked loop

### Enum rcu\_clock\_freq\_enum

**Table 3-574. Enum rcu\_clock\_freq\_enum**

enum name	Function description
CK_SYS	system clock frequency
CK_AHB	AHB clock frequency
CK_APB1	APB1 clock frequency
CK_APB2	APB2 clock frequency

### rcu\_deinit

The description of rcu\_deinit is shown as below:

**Table 3-575. Function rcu\_deinit**

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

Example:

```
/* reset RCU */
rcu_deinit();
```

### rcu\_periph\_clock\_enable

The description of rcu\_periph\_clock\_enable is shown as below:

Table 3-576. Function rcu\_periph\_clock\_enable

Function name	rcu_periph_clock_enable
Function prototype	void rcu_periph_clock_enable(rcu_periph_enum periph);
Function descriptions	enable the peripherals clock
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to <a href="#">Table 3-566. Enum rcu_periph_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the USART0 clock */
rcu_periph_clock_enable(RCU_USART0);
```

### rcu\_periph\_clock\_disable

The description of rcu\_periph\_clock\_disable is shown as below:

Table 3-577. Function rcu\_periph\_clock\_disable

Function name	rcu_periph_clock_disable
Function prototype	void rcu_periph_clock_disable(rcu_periph_enum periph);
Function descriptions	disable the peripherals clock
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to <a href="#">Table 3-566. Enum rcu_periph_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the USART0 clock */
rcu_periph_clock_disable(RCU_USART0);
```

### rcu\_periph\_clock\_sleep\_enable

The description of rcu\_periph\_clock\_sleep\_enable is shown as below:

**Table 3-578. Function rcu\_periph\_clock\_sleep\_enable**

<b>Function name</b>	rcu_periph_clock_sleep_enable
<b>Function prototype</b>	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
<b>Function descriptions</b>	enable the peripherals clock when in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <a href="#">错误!未找到引用源。</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the CRC clock when in sleep mode */
rcu_periph_clock_sleep_enable(RCU_CRC_SLP);
```

### rcu\_periph\_clock\_sleep\_disable

The description of rcu\_periph\_clock\_sleep\_disable is shown as below:

**Table 3-579. Function rcu\_periph\_clock\_sleep\_disable**

<b>Function name</b>	rcu_periph_clock_sleep_disable
<b>Function prototype</b>	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);
<b>Function descriptions</b>	disable the peripherals clock when in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
periph	RCU peripherals, refer to <a href="#">Table 3-567. Enum rcu_periph_sleep_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CRC clock when in sleep mode */
```

```
rcu_periph_clock_sleep_disable(RCU_CRC_SLP);
```

### rcu\_periph\_reset\_enable

The description of rcu\_periph\_reset\_enable is shown as below:

**Table 3-580. Function rcu\_periph\_reset\_enable**

Function name	rcu_periph_reset_enable
Function prototype	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
Function descriptions	enable the peripherals reset
Precondition	-
The called functions	-
Input parameter{in}	
periph_reset	RCU peripherals reset, refer to <a href="#">Table 3-568. Enum rcu_periph_reset_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 reset */
```

```
rcu_periph_reset_enable(RCU_SPI0RST);
```

### rcu\_periph\_reset\_disable

The description of rcu\_periph\_reset\_disable is shown as below:

Table 3-581. Function rcu\_periph\_reset\_disable

Function name	rcu_periph_reset_disable
Function prototype	void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);
Function descriptions	disable the peripheral reset
Precondition	-
The called functions	-
Input parameter{in}	
periph_reset	RCU peripherals reset, refer to <a href="#">Table 3-568. Enum rcu_periph_reset_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 reset */
```

```
rcu_periph_reset_disable(RCU_SPI0RST);
```

### rcu\_bkp\_reset\_enable

The description of rcu\_bkp\_reset\_enable is shown as below:

Table 3-582. Function rcu\_bkp\_reset\_enable

Function name	rcu_bkp_reset_enable
Function prototype	void rcu_bkp_reset_enable(void);
Function descriptions	enable the BKP domain reset
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* reset the BKP domain */
```

```
rcu_bkp_reset_enable();
```

### rcu\_bkp\_reset\_disable

The description of rcu\_bkp\_reset\_disable is shown as below:

**Table 3-583. 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 3-584. Function rcu\_system\_clock\_source\_config**

<b>Function name</b>	rcu_system_clock_source_config
<b>Function prototype</b>	void rcu_system_clock_source_config(uint32_t ck_sys);
<b>Function descriptions</b>	configure the system clock source
<b>Precondition</b>	-



<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_sys</b>	system clock source select
<i>RCU_CKSYSSRC_IRC16M</i>	select CK_IRC16M as the CK_SYS source
<i>RCU_CKSYSSRC_HXTAL</i>	select CK_HXTAL as the CK_SYS source
<i>RCU_CKSYSSRC_PLLP</i>	select CK_PLLP as the CK_SYS source
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CK_HXTAL as the CK_SYS source */
rcu_system_clock_source_config(RCU_CKSYSSRC_HXTAL);
```

### rcu\_system\_clock\_source\_get

The description of rcu\_system\_clock\_source\_get is shown as below:

**Table 3-585. 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_IRC16M</i>	CK_IRC16M is selected as the CK_SYS source
<i>RCU_SCSS_HXTAL</i>	CK_HXTAL is selected as the CK_SYS source
<i>RCU_SCSS_PLLP</i>	CK_PLLP 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 3-586. 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, (x=1, 2, 4, 8, 16, 64, 128, 256, 512)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure CK_SYS/128 */

rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

### rcu\_apb1\_clock\_config

The description of rcu\_apb1\_clock\_config is shown as below:

Table 3-587. Function rcu\_apb1\_clock\_config

Function name	rcu_apb1_clock_config
Function prototype	void rcu_apb1_clock_config(uint32_t ck_apb1);
Function descriptions	configure the APB1 clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
ck_apb1	APB1 clock prescaler selection
RCU_APB1_CKAHB_D IV1	select CK_AHB as CK_APB1
RCU_APB1_CKAHB_D IV2	select CK_AHB/2 as CK_APB1
RCU_APB1_CKAHB_D IV4	select CK_AHB/4 as CK_APB1
RCU_APB1_CKAHB_D IV8	select CK_AHB/8 as CK_APB1
RCU_APB1_CKAHB_D IV16	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 3-588. Function rcu\_apb2\_clock\_config

Function name	rcu_apb2_clock_config
Function prototype	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\_ckout0\_config

The description of rcu\_ckout0\_config is shown as below:

**Table 3-589. Function rcu\_ckout0\_config**

<b>Function name</b>	rcu_ckout0_config
<b>Function prototype</b>	void rcu_ckout0_config(uint32_t ckout0_src, uint32_t ckout0_div);
<b>Function descriptions</b>	configure the CK_OUT0 clock source
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>ckout0_src</b>	CK_OUT0 clock source selection
<i>RCU_CKOUT0SRC_IRC16M</i>	select IRC16M clock
<i>RCU_CKOUT0SRC_LXTAL</i>	select LXTAL clock
<i>RCU_CKOUT0SRC_HXTAL</i>	select HXTAL clock
<i>RCU_CKOUT0SRC_CKPLL_PLLP</i>	select PLLP clock
Input parameter{in}	
<b>ckout0_div</b>	CK_OUT0 divider
<i>RCU_CKOUT0_DIVx(x=1,2,3,4,5)</i>	CK_OUT0 is divided by x
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the HXTAL as CK_OUT0 clock source */
```

```
rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL, RCU_CKOUT0_DIV1);
```

### rcu\_ckout1\_config

The description of rcu\_ckout1\_config is shown as below:

**Table 3-590. Function rcu\_ckout1\_config**

<b>Function name</b>	rcu_ckout1_config
<b>Function prototype</b>	void rcu_ckout1_config(uint32_t ckout1_src, uint32_t ckout1_div);
<b>Function descriptions</b>	configure the CK_OUT1 clock source and divider
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	

<b>ckout1_src</b>	CK_OUT1 clock source selection
<i>RCU_CKOUT1SRC_SYSTEMCLOCK</i>	select system clock
<i>RCU_CKOUT1SRC_PLI2SR</i>	select PLLI2SR clock
<i>RCU_CKOUT1SRC_HXTAL</i>	select HXTAL clock
<i>RCU_CKOUT1SRC_PLLP</i>	select PLLP clock
<b>Input parameter{in}</b>	
<b>ckout1_div</b>	CK_OUT1 divider
<i>RCU_CKOUT1_DIVx(x=1,2,3,4,5)</i>	CK_OUT1 is divided by x
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the HXTAL as CK_OUT1 clock source */
```

```
rcu_ckout1_config(RCU_CKOUT1SRC_HXTAL, RCU_CKOUT1_DIV1);
```

### rcu\_pll\_config

The description of rcu\_pll\_config is shown as below:

**Table 3-591. Function rcu\_pll\_config**

<b>Function name</b>	rcu_pll_config
<b>Function prototype</b>	ErrStatus rcu_pll_config(uint32_t pll_src, uint32_t pll_psc, uint32_t pll_n, uint32_t pll_p, uint32_t pll_q);
<b>Function descriptions</b>	configure the main PLL clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>pll_src</b>	PLL clock source selection
<i>RCU_PLLSRC_IRC16M</i>	IRC16M clock is selected as source clock of PLL, PLLSAI, PLLI2S
<i>RCU_PLLSRC_HXTAL</i>	HXTAL is selected as source clock of PLL, PLLSAI, PLLI2S
<b>Input parameter{in}</b>	
<b>pll_psc</b>	the PLL VCO source clock prescaler
<i>uint32_t</i>	2~63
<b>Input parameter{in}</b>	
<b>pll_n</b>	the PLL VCO clock multi factor
<i>uint32_t</i>	64~500
<b>Input parameter{in}</b>	
<b>pll_p</b>	the PLLP output frequency division factor from PLL VCO clock
<i>uint32_t</i>	2,4,6,8
<b>Input parameter{in}</b>	
<b>pll_q</b>	the PLL Q output frequency division factor from PLL VCO clock
<i>uint32_t</i>	2~15
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* Configure the main PLL, PSC = 8, PLL_N = 240, PLL_P = 2, PLL_Q = 5 */
```

```
rcu_pll_config(RCU_PLLSRC_HXTAL,8,240,2,5);
```

### rcu\_plli2s\_config

The description of rcu\_plli2s\_config is shown as below:

**Table 3-592. Function rcu\_plli2s\_config**

<b>Function name</b>	rcu_plli2s_config
<b>Function prototype</b>	ErrStatus rcu_plli2s_config(uint32_t plli2s_n, uint32_t plli2s_r);
<b>Function descriptions</b>	configure the PLLI2S clock

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>plli2s_n</b>	the PLLI2S VCO clock multi factor
<i>uint32_t</i>	50~500
<b>Input parameter{in}</b>	
<b>plli2s_r</b>	the PLLI2S R output frequency division factor from PLLI2S VCO clock
<i>uint32_t</i>	2~7
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* configure the PLLI2S n = 100, PLLI2S r =2*/
```

```
rcu_plli2s_config (100, 2);
```

### rcu\_pllsai\_config

The description of rcu\_pllsai\_config is shown as below:

**Table 3-593. Function rcu\_pllsai\_config**

<b>Function name</b>	rcu_pllsai_config
<b>Function prototype</b>	ErrStatus rcu_pllsai_config(uint32_t pllsai_n, uint32_t pllsai_p, uint32_t pllsai_r);
<b>Function descriptions</b>	configure the PLLSAI clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pllsai_n</b>	the PLLSAI VCO clock multi factor
<i>uint32_t</i>	50~500
<b>Input parameter{in}</b>	
<b>pllsai_p</b>	the PLLSAI P output frequency division factor from PLL VCO clock



<i>uint32_t</i>	2,4,6,8
<b>Input parameter{in}</b>	
<b>pllsai_r</b>	the PLLSAI R output frequency division factor from PLL VCO clock
<i>uint32_t</i>	2~7
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* configure the PLLSAI n = 100, PLLSAI p = 2, PLLSAI r = 2 */
```

```
rcu_pllsai_config (100, 2,2);
```

### rcu\_rtc\_clock\_config

The description of rcu\_rtc\_clock\_config is shown as below:

**Table 3-594. Function rcu\_rtc\_clock\_config**

<b>Function name</b>	rcu_rtc_clock_config
<b>Function prototype</b>	void rcu_rtc_clock_config(uint32_t rtc_clock_source);
<b>Function descriptions</b>	configure the RTC clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_clock_source</b>	RTC clock source selection
<i>RCU_RTCSRC_NONE</i>	no clock selected
<i>RCU_RTCSRC_LXTAL</i>	select CK_LXTAL as RTC source clock
<i>RCU_RTCSRC_IRC32K</i>	select CK_IRC32K as RTC source clock
<i>RCU_RTCSRC_HXTAL_DIV_RTCDIV</i>	select CK_HXTAL or RTCDIV as RTC source clock
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure the RTC clock source selection */
rcu_rtc_clock_config(RCU_RTCSRC_IRC32K);
```

### rcu\_rtc\_div\_config

The description of rcu\_rtc\_div\_config is shown as below:

**Table 3-595. Function rcu\_rtc\_div\_config**

Function name	rcu_rtc_div_config
Function prototype	void rcu_rtc_div_config(uint32_t rtc_div);
Function descriptions	configure the frequency division of RTC clock when HXTAL was selected as its clock source
Precondition	-
The called functions	-
Input parameter{in}	
rtc_div	RTC clock frequency division
RCU_RTC_HXTAL_NO NE	no clock for RTC
RCU_RTC_HXTAL_DI Vx	RTCDIV clock select CK_HXTAL/x, x = 2,3,...31
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the frequency division of RTC clock when HXTAL was selected as its clock source */
rcu_rtc_div_config(RCU_RTC_HXTAL_DIV6);
```

### rcu\_i2s\_clock\_config

The description of rcu\_i2s\_clock\_config is shown as below:

Table 3-596. Function rcu\_i2s\_clock\_config

Function name	rcu_i2s_clock_config
Function prototype	void rcu_i2s_clock_config(uint32_t i2s_clock_source);
Function descriptions	configure the I2S clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
i2s_clock_source	I2S clock source selection
RCU_I2SSRC_PLLI2S	CK_PLLI2S selected as I2S source clock
RCU_I2SSRC_I2S_CKIN	external i2s_ckin pin selected as I2S source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the I2S clock source selection */
rcu_i2s_clock_config(RCU_I2SSRC_PLLI2S);
```

### rcu\_ck48m\_clock\_config

The description of rcu\_ck48m\_clock\_config is shown as below:

Table 3-597. Function rcu\_ck48m\_clock\_config

Function name	rcu_ck48m_clock_config
Function prototype	void rcu_ck48m_clock_config(uint32_t ck48m_clock_source);
Function descriptions	configure the CK48M clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
ck48m_clock_source	CK48M clock source selection
RCU_CK48MSRC_PLL	CK_PLL48M selected as CK48M source clock

48M	
RCU_CK48MSRC_IRC 48M	CK_IRC48M selected as CK48M source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the CK48M clock source selection */
```

```
rcu_ck48m_clock_config (RCU_CK48MSRC_IRC48M);
```

### rcu\_pll48m\_clock\_config

The description of rcu\_pll48m\_clock\_config is shown as below:

**Table 3-598. Function rcu\_pll48m\_clock\_config**

Function name	rcu_pll48m_clock_config
Function prototype	void rcu_pll48m_clock_config(uint32_t pll48m_clock_source);
Function descriptions	configure the PLL48M clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
pll48m_clock_source	PLL48M clock source selection
RCU_PLL48MSRC_PL LQ	CK_PLLQ selected as PLL48M source clock
RCU_PLL48MSRC_PL LSAIP	CK_PLLSAIP selected as PLL48M source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLL48M clock selection */
```

```
rcu_pll48m_clock_config(RCU_PLL48MSRC_PLLQ);
```

### rcu\_timer\_clock\_prescaler\_config

The description of rcu\_timer\_clock\_prescaler\_config is shown as below:

**Table 3-599. Function rcu\_timer\_clock\_prescaler\_config**

<b>Function name</b>	rcu_timer_clock_prescaler_config
<b>Function prototype</b>	void rcu_timer_clock_prescaler_config(uint32_t timer_clock_prescaler);
<b>Function descriptions</b>	configure the TIMER clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_clock_prescaler</b>	TIMER时钟源选择
<i>RCU_TIMER_PSC_MU_L2</i>	If CK_APBx = CK_AHB or CK_APBx = CK_AHB/2, CK_TIMERx = CK_AHB, else CK_TIMERx = 2 x CK_APBx
<i>RCU_TIMER_PSC_MU_L4</i>	If CK_APBx = CK_AHB or CK_APBx = CK_AHB/2 or CK_APBx = CK_AHB/4, CK_TIMERx = CK_AHB, else CK_TIMERx = 4 x CK_APBx
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER clock source */
```

```
rcu_timer_clock_prescaler_config(RCU_TIMER_PSC_MUL4);
```

### rcu\_tli\_clock\_div\_config

The description of rcu\_tli\_clock\_div\_config is shown as below:

**Table 3-600. Function rcu\_tli\_clock\_div\_config**

<b>Function name</b>	rcu_tli_clock_div_config
<b>Function prototype</b>	void rcu_tli_clock_div_config(uint32_t pllsai_r_div);
<b>Function descriptions</b>	configure the PLLSAIR divider used as input of TLI
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>pllai_r_div</b>	PLLSAIR divider used as input of TLI
<i>RCU_PLLSAIR_DIV2</i>	PLLSAIR/2
<i>RCU_PLLSAIR_DIV4</i>	PLLSAIR/4
<i>RCU_PLLSAIR_DIV8</i>	PLLSAIR/8
<i>RCU_PLLSAIR_DIV16</i>	PLLSAIR/16
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the TLI prescaler factor from PLLSAIR clock */
```

```
rcu_tli_clock_div_config (RCU_PLLSAIR_DIV4);
```

### rcu\_flag\_get

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

**Table 3-601. Function rcu\_flag\_get**

<b>Function name</b>	rcu_flag_get
<b>Function prototype</b>	FlagStatus rcu_flag_get(rcu_flag_enum flag);
<b>Function descriptions</b>	get the clock stabilization and peripheral reset flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	the clock stabilization and peripheral reset flags, refer to <a href="#">Table 3-569. Enum rcu_flag_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the clock stabilization flag */
if(RESET != rcu_flag_get(RCU_FLAG_LXTALSTB)){
}
```

### rcu\_all\_reset\_flag\_clear

The description of rcu\_all\_reset\_flag\_clear is shown as below:

**Table 3-602. Function rcu\_all\_reset\_flag\_clear**

<b>Function name</b>	rcu_all_reset_flag_clear
<b>Function prototype</b>	void rcu_all_reset_flag_clear(void);
<b>Function descriptions</b>	clear all the reset flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear all the reset flag */
rcu_all_reset_flag_clear();
```

### rcu\_interrupt\_flag\_get

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

**Table 3-603. 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 stabilization interrupt and ckm flags
<b>Precondition</b>	-

The called functions	-
Input parameter{in}	
int_flag	interrupt and ckm flags, refer to <a href="#">Table 3-570. Enum rcu_int_flag_enum</a>
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the clock stabilization interrupt flag */
if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){
}
```

### rcu\_interrupt\_flag\_clear

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

**Table 3-604. Function rcu\_interrupt\_flag\_clear**

Function name	rcu_interrupt_flag_clear
Function prototype	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag_clear)
Function descriptions	clear the interrupt flags
Precondition	-
The called functions	-
Input parameter{in}	
int_flag_clear	clock stabilization and stuck interrupt flags clear, refer to <a href="#">Table 3-571. Enum rcu_int_flag_clear_enum</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the interrupt HXTAL stabilization interrupt flag */
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```



## rcu\_interrupt\_enable

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

**Table 3-605. Function rcu\_interrupt\_enable**

<b>Function name</b>	rcu_interrupt_enable
<b>Function prototype</b>	void rcu_interrupt_enable(rcu_int_enum interrupt);
<b>Function descriptions</b>	enable the stabilization interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>stab_int</b>	clock stabilization interrupt, refer to <a href="#">Table 3-572. Enum rcu_int_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the HXTAL stabilization interrupt */
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

## rcu\_interrupt\_disable

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

**Table 3-606. Function rcu\_interrupt\_disable**

<b>Function name</b>	rcu_interrupt_disable
<b>Function prototype</b>	void rcu_interrupt_disable(rcu_int_enum interrupt);
<b>Function descriptions</b>	disable the stabilization interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>stab_int</b>	clock stabilization interrupt, refer to <a href="#">Table 3-572. Enum rcu_int_enum</a>
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* disable the HXTAL stabilization interrupt */
```

```
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

### rcu\_lxtal\_drive\_capability\_config

The description of rcu\_lxtal\_drive\_capability\_config is shown as below:

**Table 3-607. Function rcu\_lxtal\_drive\_capability\_config**

Function name	rcu_lxtal_drive_capability_config
Function prototype	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
Function descriptions	configure the LXTAL drive capability
Precondition	-
The called functions	-
Input parameter{in}	
lxtal_dricap	drive capability of LXTAL
RCU_LXTALDRI_LOWER_DRIVE	lower driving capability
RCU_LXTALDRI_HIGHER_DRIVE	higher driving capability
Output parameter{out}	
-	-
Return value	
-	-

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 3-608. Function rcu\_osci\_stab\_wait

Function name	rcu_osci_stab_wait
Function prototype	ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci);
Function descriptions	wait for oscillator stabilization flags is SET or oscillator startup is timeout
Precondition	-
The called functions	rcu_flag_get
Input parameter{in}	
osci	oscillator types, refer to <a href="#">Table 3-573. Enum rcu_osci_type_enum</a>
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* wait for oscillator stabilization flag */
if(SUCCESS == rcu_osci_stab_wait(RCU_HXTAL)){
}
```

### rcu\_osci\_on

The description of rcu\_osci\_on is shown as below:

Table 3-609. Function rcu\_osci\_on

Function name	rcu_osci_on
Function prototype	void rcu_osci_on(rcu_osci_type_enum osci);
Function descriptions	turn on the oscillator
Precondition	-
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to <a href="#">Table 3-573. Enum rcu_osci_type_enum</a>
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* turn on the high speed crystal oscillator */
```

```
rcu_osci_on(RCU_HXTAL);
```

### rcu\_osci\_off

The description of rcu\_osci\_off is shown as below:

**Table 3-610. Function rcu\_osci\_off**

<b>Function name</b>	rcu_osci_off
<b>Function prototype</b>	void rcu_osci_off(rcu_osci_type_enum osci);
<b>Function descriptions</b>	turn off the oscillator
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <a href="#">Table 3-573. Enum rcu_osci_type_enum</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn off the high speed crystal oscillator */
```

```
rcu_osci_off(RCU_HXTAL);
```

### rcu\_osci\_bypass\_mode\_enable

The description of rcu\_osci\_bypass\_mode\_enable is shown as below:

**Table 3-611. Function rcu\_osci\_bypass\_mode\_enable**

<b>Function name</b>	rcu_osci_bypass_mode_enable
<b>Function prototype</b>	void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);
<b>Function descriptions</b>	enable the oscillator bypass mode
<b>Precondition</b>	HXTALEN or LXTALEN must be reset before it

The called functions	-
Input parameter{in}	
osci	oscillator types, refer to <a href="#">Table 3-573. Enum rcu_osci_type_enum</a>
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

### rcu\_osci\_bypass\_mode\_disable

The description of rcu\_osci\_bypass\_mode\_disable is shown as below:

**Table 3-612. Function rcu\_osci\_bypass\_mode\_disable**

Function name	rcu_osci_bypass_mode_disable
Function prototype	void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci);
Function descriptions	disable the oscillator bypass mode
Precondition	HXTALEN or LXTALEN must be reset before it
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to <a href="#">Table 3-573. Enum rcu_osci_type_enum</a>
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the high speed crystal oscillator bypass mode */
rcu_osci_bypass_mode_disable(RCU_HXTAL);
```

### rcu\_hxtal\_clock\_monitor\_enable

The description of rcu\_hxtal\_clock\_monitor\_enable is shown as below:

**Table 3-613. 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 3-614. 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>	-

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

Example:

```
/* disable the HXTAL clock monitor */
```

```
rcu_hxtal_clock_monitor_disable();
```

### rcu\_irc16m\_adjust\_value\_set

The description of rcu\_irc16m\_adjust\_value\_set is shown as below:

**Table 3-615. Function rcu\_irc16m\_adjust\_value\_set**

Function name	rcu_irc16m_adjust_value_set
Function prototype	void rcu_irc16m_adjust_value_set(uint8_t irc16m_adjval);
Function descriptions	set the IRC16M adjust value
Precondition	-
The called functions	-
Input parameter{in}	
irc16m_adjval	IRC16M adjust value, must be between 0 and 0x1F
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the IRC16M adjust value */
```

```
rcu_irc16m_adjust_value_set(0x10);
```

### rcu\_spread\_spectrum\_config

The description of rcu\_spread\_spectrum\_config is shown as below:

Table 3-616. Function rcu\_spread\_spectrum\_config

Function name	rcu_spread_spectrum_config
Function prototype	void rcu_spread_spectrum_config(uint32_t spread_spectrum_type, uint32_t modstep, uint32_t modcnt)
Function descriptions	configure the spread spectrum modulation for the main PLL clock
Precondition	-
The called functions	-
Input parameter{in}	
spread_spectrum_type	PLL spread spectrum modulation type select
RCU_SS_TYPE_CENTER	center spread type is selected
RCU_SS_TYPE_DOWN	down spread type is selected
Input parameter{in}	
modstep	configure PLL spread spectrum modulation profile amplitude
uint32_t	0 ~ 0x7FFF, The following criteria must be met: MODSTEP*MODCNT <=2 <sup>15</sup> -1
Input parameter{in}	
modcnt	configure PLL spread spectrum modulation profile frequency
uint32_t	0 ~ 0x1FFF, The following criteria must be met: MODSTEP*MODCNT <=2 <sup>15</sup> -1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure PLL spread_spectrum */
```

```
rcu_spread_spectrum_config (RCU_SS_TYPE_CENTER, 0x0F,0x0F);
```

### rcu\_spread\_spectrum\_enable

The description of rcu\_spread\_spectrum\_enable is shown as below:



Table 3-617. Function rcu\_spread\_spectrum\_enable

Function name	rcu_spread_spectrum_enable
Function prototype	void rcu_spread_spectrum_enable(void);
Function descriptions	enable the PLL spread spectrum modulation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the PLL spread spectrum modulation */
rcu_spread_spectrum_enable ();
```

### rcu\_spread\_spectrum\_disable

The description of rcu\_spread\_spectrum\_disable is shown as below:

Table 3-618. Function rcu\_spread\_spectrum\_disable

Function name	rcu_spread_spectrum_disable
Function prototype	void rcu_spread_spectrum_disable(void);
Function descriptions	disable the HXTAL clock monitor
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the PLL spread spectrum modulation */
rcu_spread_spectrum_disable();
```

### rcu\_voltage\_key\_unlock

The description of rcu\_voltage\_key\_unlock is shown as below:

**Table 3-619. Function rcu\_voltage\_key\_unlock**

Function name	rcu_voltage_key_unlock
Function prototype	void rcu_voltage_key_unlock (void);
Function descriptions	unlock the voltage key
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock the voltage key register */
rcu_voltage_key_unlock ();
```

### rcu\_deepsleep\_voltage\_set

The description of rcu\_deepsleep\_voltage\_set is shown as below:

**Table 3-620. Function rcu\_deepsleep\_voltage\_set**

Function name	rcu_deepsleep_voltage_set
Function prototype	void rcu_deepsleep_voltage_set(uint32_t dsvol);
Function descriptions	set the deep-sleep mode voltage value
Precondition	-
The called functions	-

Input parameter{in}	
<b>dsvol</b>	deep sleep mode voltage
<i>RCU_DEEPSLEEP_V_0</i>	the core voltage is default value in deep-sleep mode
<i>RCU_DEEPSLEEP_V_1</i>	the core voltage is (default value-0.1)V in deep-sleep mode(customers are not recommended to use it)
<i>RCU_DEEPSLEEP_V_2</i>	the core voltage is (default value-0.2)V in deep-sleep mode(customers are not recommended to use it)
<i>RCU_DEEPSLEEP_V_3</i>	the core voltage is (default value-0.3)V in deep-sleep mode(customers are not recommended to use it)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the deep-sleep mode voltage */
```

```
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1);
```

### rcu\_clock\_freq\_get

The description of rcu\_clock\_freq\_get is shown as below:

**Table 3-621. Function rcu\_clock\_freq\_get**

<b>Function name</b>	rcu_clock_freq_get
<b>Function prototype</b>	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
<b>Function descriptions</b>	get the system clock, bus clock frequency
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>clock</b>	the clock frequency which to get, refer to <a href="#">Table 3-574. Enum rcu_clock_freq_enum</a>
Output parameter{out}	
-	-

Return value	
<b>ck_freq</b>	clock frequency of system, AHB, APB1, APB2

Example:

```
uint32_t temp_freq;

/* get the system clock frequency */

temp_freq = rcu_clock_freq_get(CK_SYS);
```

## 3.22. 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 3-622. RTC Registers**

Registers	Descriptions
RTC_TIME	RTC time of day register
RTC_DATE	RTC date register
RTC_CTL	RTC control register
RTC_STAT	RTC status register
RTC_PSC	RTC time prescaler register
RTC_WUT	RTC wakeup timer register
RTC_COSC	RTC coarse calibration register
RTC_ALRM0TD	RTC alarm 0 time and date register
RTC_ALRM1TD	RTC alarm 1 time and date register
RTC_WPK	RTC write protection key register
RTC_SS	RTC sub second register
RTC_SHIFTCTL	RTC shift function control register
RTC_TTS	RTC time of timestamp register

Registers	Descriptions
RTC_DTS	RTC date of timestamp register
RTC_SSTS	RTC sub second of timestamp register
RTC_HRFC	RTC high resolution frequency compensation register
RTC_TAMP	RTC tamper register
RTC_ALRM0SS	RTC alarm 0 sub second register
RTC_ALRM1SS	RTC alarm 1 sub second register
RTC_BKPx(x = 0, 1, 2, ..., 18, 19)	RTC backup register

### 3.22.2. Descriptions of Peripheral functions

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

**Table 3-623.RTC firmware function**

Function name	Function description
rtc_deinit	reset most of the RTC registers
rtc_init	initialize RTC registers
rtc_init_mode_enter	enter RTC init mode
rtc_init_mode_exit	exit RTC init mode
rtc_register_sync_wait	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
rtc_current_time_get	get current time and date
rtc_subsecond_get	get current subsecond value
rtc_alarm_config	configure RTC alarm
rtc_alarm_subsecond_config	configure subsecond of RTC alarm
rtc_alarm_get	get RTC alarm
rtc_alarm_subsecond_get	get RTC alarm subsecond
rtc_alarm_enable	enable RTC alarm
rtc_alarm_disable	disable RTC alarm
rtc_timestamp_enable	enable RTC time-stamp

Function name	Function description
rtc_timestamp_disable	disable RTC time-stamp
rtc_timestamp_get	get RTC timestamp time and date
rtc_timestamp_subsecond_get	get RTC time-stamp subsecond
rtc_timestamp_pin_map	RTC time-stamp pin map
rtc_tamper_enable	enable RTC tamper
rtc_tamper_disable	disable RTC tamper
rtc_tamper0_pin_map	RTC tamper0 pin map
rtc_interrupt_enable	enable specified RTC interrupt
rtc_interrupt_disable	disble specified RTC interrupt
rtc_flag_get	check specified flag
rtc_flag_clear	clear specified flag
rtc_alter_output_config	configure RTC alternate output source
rtc_calibration_output_config	configure RTC calibration output source
rtc_hour_adjust	ajust the daylight saving time by adding or substracting one hour from the current time
rtc_second_adjust	ajust RTC second or subsecond value of current time
rtc_bypass_shadow_enable	enable RTC bypass shadow registers function
rtc_bypass_shadow_disable	disable RTC bypass shadow registers function
rtc_refclock_detection_enable	enable RTC reference clock detection function
rtc_refclock_detection_disable	disable RTC reference clock detection function
rtc_wakeup_enable	enable RTC wakeup timer
rtc_wakeup_disable	disable RTC wakeup timer
rtc_wakeup_clock_set	set auto wakeup timer clock
rtc_wakeup_timer_set	set auto wakeup timer value
rtc_wakeup_timer_get	get auto wakeup timer value
rtc_smooth_calibration_config	configure RTC smooth calibration
rtc_coarse_calibration_enable	enable RTC coarse calibration
rtc_coarse_calibration_disable	disable RTC coarse calibration

Function name	Function description
rtc_coarse_calibration_config	configure RTC coarse calibration direction and step

### Structure rtc\_parameter\_struct

Table 3-624. Struct rtc\_parameter\_struct

Member name	Function description
year	RTC year value: 0x0 - 0x99(BCD format)
month	RTC month value
date	RTC date value: 0x1 - 0x31(BCD format)
day_of_week	RTC weekday value
hour	RTC hour value
minute	RTC minute value: 0x0 - 0x59(BCD format)
second	RTC second value: 0x0 - 0x59(BCD format)
factor_asyn	RTC asynchronous prescaler value: 0x0 - 0x7F
factor_syn	RTC synchronous prescaler value: 0x0 - 0x7FFF
am_pm	RTC AM/PM value
display_format	RTC time notation

### Structure rtc\_alarm\_struct

Table 3-625. Struct rtc\_alarm\_struct

Member name	Function description
alarm_mask	RTC alarm mask
weekday_or_date	specify RTC alarm is on date or weekday
alarm_day	RTC alarm date or weekday value
alarm_hour	RTC alarm hour value
alarm_minute	RTC alarm minute value: 0x0 - 0x59(BCD format)
alarm_second	RTC alarm second value: 0x0 - 0x59(BCD format)
am_pm	RTC alarm AM/PM value

## Structure rtc\_timestamp\_struct

**Table 3-626. Struct rtc\_timestamp\_struct**

Member name	Function description
timestamp_month	RTC time-stamp month value
timestamp_date	RTC time-stamp date value: 0x1 - 0x31(BCD format)
timestamp_day	RTC time-stamp weekday value
timestamp_hour	RTC time-stamp hour value
timestamp_minute	RTC time-stamp minute value: 0x0 - 0x59(BCD format)
timestamp_second	RTC time-stamp second value: 0x0 - 0x59(BCD format)
am_pm	RTC time-stamp AM/PM value

## Structure rtc\_tamper\_struct

**Table 3-627. Struct rtc\_tamper\_struct**

Member name	Function description
tamper_source	RTC tamper source
tamper_trigger	RTC tamper trigger
tamper_filter	RTC tamper consecutive samples needed during a voltage level detection
tamper_sample_frequency	RTC tamper sampling frequency during a voltage level detection
tamper_precharge_enable	RTC tamper precharge feature during a voltage level detection
tamper_precharge_time	RTC tamper precharge duration if precharge feature is enabled
tamper_with_timestamp	RTC tamper time-stamp feature

## rtc\_deinit

The description of rtc\_deinit is shown as below:

**Table 3-628. Function rtc\_deinit**

Function name	rtc_deinit
Function prototype	ErrStatus rtc_deinit(void);
Function descriptions	reset most of the RTC registers



<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable/ rcu_periph_reset_disable -
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* reset most of the RTC registers*/
```

```
ErrStatus error_status = rtc_deinit();
```

### rtc\_init

The description of rtc\_init is shown as below:

**Table 3-629. Function rtc\_init**

<b>Function name</b>	rtc_init
<b>Function prototype</b>	ErrStatus rtc_init(rtc_parameter_struct* rtc_initpara_struct);
<b>Function descriptions</b>	initialize RTC registers
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_initpara_struct</b>	the structure members can refer to members of the structure <a href="#">Table 3-624. Struct rtc_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* reset most of the RTC registers*/
```

```
rtc_parameter_struct rtc_initpara_struct;
```

```
ErrStatus error_status = rtc_init (&rtc_initpara_struct);
```

### rtc\_init\_mode\_enter

The description of rtc\_init\_mode\_enter is shown as below:

**Table 3-630. Function rtc\_init\_mode\_enter**

<b>Function name</b>	rtc_init_mode_enter
<b>Function prototype</b>	ErrStatus rtc_init_mode_enter(void);
<b>Function descriptions</b>	enter RTC init mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/*enter RTC init mode*/
```

```
ErrStatus error_status = rtc_init_mode_enter ();
```

### rtc\_init\_mode\_exit

The description of rtc\_init\_mode\_exit is shown as below:

**Table 3-631. Function rtc\_init\_mode\_exit**

<b>Function name</b>	rtc_init_mode_exit
<b>Function prototype</b>	void rtc_init_mode_exit(void);
<b>Function descriptions</b>	exit RTC init mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*exit RTC init mode*/
```

```
rtc_init_mode_exit ();
```

### rtc\_register\_sync\_wait

The description of rtc\_register\_sync\_wait is shown as below:

**Table 3-632. Function rtc\_register\_sync\_wait**

Function name	rtc_register_sync_wait
Function prototype	ErrStatus rtc_register_sync_wait(void);
Function descriptions	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/*wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated*/
```

```
ErrStatus error_status = rtc_register_sync_wait ();
```

### rtc\_current\_time\_get

The description of rtc\_current\_time\_get is shown as below:

Table 3-633. Function rtc\_current\_time\_get

Function name	rtc_current_time_get
Function prototype	void rtc_current_time_get(rtc_parameter_struct* rtc_initpara_struct);
Function descriptions	get current time and date
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
rtc_initpara_struct	the structure members can refer to members of the structure <a href="#">Table 3-624.</a> <a href="#">Struct rtc_parameter_struct</a>
Return value	
-	-

Example:

```
/*get current time and date*/
```

```
rtc_parameter_struct rtc_initpara_struct;
```

```
rtc_current_time_get (&rtc_initpara_struct);
```

### rtc\_subsecond\_get

The description of rtc\_subsecond\_get is shown as below:

Table 3-634. Function rtc\_subsecond\_get

Function name	rtc_subsecond_get
Function prototype	uint32_t rtc_subsecond_get(void);
Function descriptions	get current subsecond value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
uint32_t	current subsecond value(0x00-0xFFFF)

Example:

```
/*get current subsecond value*/
```

```
uint32_t sub_second = rtc_subsecond_get();
```

### rtc\_alarm\_config

The description of rtc\_alarm\_config is shown as below:

**Table 3-635. Function rtc\_alarm\_config**

Function name	rtc_alarm_config
Function prototype	void rtc_alarm_config(uint8_t rtc_alarm, rtc_alarm_struct* rtc_alarm_time);
Function descriptions	configure RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm0 or alarm1
RTC_ALARM0	alarm0
RTC_ALARM1	alarm1
Input parameter{in}	
rtc_alarm_time	the structure members can refer to members of the structure <a href="#">Table 3-625. Struct rtc_alarm_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure RTC alarm0*/
```

```
rtc_alarm_struct rtc_alarm_time;
```

```
rtc_alarm_config(RTC_ALARM0, &rtc_alarm_time);
```

## rtc\_alarm\_subsecond\_config

The description of rtc\_alarm\_subsecond\_config is shown as below:

**Table 3-636. Function rtc\_alarm\_subsecond\_config**

<b>Function name</b>	rtc_alarm_subsecond_config
<b>Function prototype</b>	void rtc_alarm_subsecond_config(uint8_t rtc_alarm, uint32_t mask_subsecond, uint32_t subsecond);
<b>Function descriptions</b>	configure subsecond of RTC alarm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_alarm</b>	select alarm0 or alarm1
<i>RTC_ALARM0</i>	alarm0
<i>RTC_ALARM1</i>	alarm1
<b>Input parameter{in}</b>	
<b>mask_subsecond</b>	alarm subsecond mask
<i>RTC_MASKSSC_0_14</i>	mask alarm subsecond configuration
<i>RTC_MASKSSC_1_14</i>	mask RTC_ALRM0SS_SSC[14:1], and RTC_ALRM0SS_SSC[0] is to be compared
<i>RTC_MASKSSC_2_14</i>	mask RTC_ALRM0SS_SSC[14:2], and RTC_ALRM0SS_SSC[1:0] is to be compared
<i>RTC_MASKSSC_3_14</i>	mask RTC_ALRM0SS_SSC[14:3], and RTC_ALRM0SS_SSC[2:0] is to be compared
<i>RTC_MASKSSC_4_14</i>	mask RTC_ALRM0SS_SSC[14:4], and RTC_ALRM0SS_SSC[3:0] is to be compared
<i>RTC_MASKSSC_5_14</i>	mask RTC_ALRM0SS_SSC[14:5], and RTC_ALRM0SS_SSC[4:0] is to be compared
<i>RTC_MASKSSC_6_14</i>	mask RTC_ALRM0SS_SSC[14:6], and RTC_ALRM0SS_SSC[5:0] is to be compared
<i>RTC_MASKSSC_7_14</i>	mask RTC_ALRM0SS_SSC[14:7], and RTC_ALRM0SS_SSC[6:0] is to be compared
<i>RTC_MASKSSC_8_14</i>	mask RTC_ALRM0SS_SSC[14:8], and RTC_ALRM0SS_SSC[7:0] is to be compared

<i>RTC_MASKSSC_9_14</i>	mask RTC_ALARM0SS_SSC[14:9], and RTC_ALARM0SS_SSC[8:0] is to be compared
<i>RTC_MASKSSC_10_14</i> 4	mask RTC_ALARM0SS_SSC[14:10], and RTC_ALARM0SS_SSC[9:0] is to be compared
<i>RTC_MASKSSC_11_14</i> 4	mask RTC_ALARM0SS_SSC[14:11], and RTC_ALARM0SS_SSC[10:0] is to be compared
<i>RTC_MASKSSC_12_14</i> 4	mask RTC_ALARM0SS_SSC[14:12], and RTC_ALARM0SS_SSC[11:0] is to be compared
<i>RTC_MASKSSC_13_14</i> 4	mask RTC_ALARM0SS_SSC[14:13], and RTC_ALARM0SS_SSC[12:0] is to be compared
<i>RTC_MASKSSC_14</i>	mask RTC_ALARM0SS_SSC[14], and RTC_ALARM0SS_SSC[13:0] is to be compared
<i>RTC_MASKSSC_NONE</i>	mask none, and RTC_ALARM0SS_SSC[14:0] is to be compared
<b>Input parameter{in}</b>	
<b>subsecond</b>	alarm subsecond value(0x000 - 0x7FFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*configure subsecond of RTC alarm0*/
```

```
rtc_alarm_subsecond_config (RTC_ALARM0, RTC_MASKSSC_9_14, 0x7FFF);
```

### rtc\_alarm\_get

The description of rtc\_alarm\_get is shown as below:

**Table 3-637. Function rtc\_alarm\_get**

<b>Function name</b>	rtc_alarm_get
<b>Function prototype</b>	void rtc_alarm_get(uint8_t rtc_alarm, rtc_alarm_struct* rtc_alarm_time);
<b>Function descriptions</b>	get RTC alarm
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>rtc_alarm</b>	select alarm0 or alarm1
<i>RTC_ALARM0</i>	alarm0
<i>RTC_ALARM1</i>	alarm1
Output parameter{out}	
<b>rtc_alarm_time</b>	the structure members can refer to members of the structure <a href="#">Table 3-625.</a> <a href="#">Struct rtc_alarm_struct</a>
Return value	
-	-

Example:

```
/*get RTC alarm0*/
rtc_alarm_struct rtc_alarm_time;
rtc_alarm_get (RTC_ALARM0, &rtc_alarm_time);
```

### rtc\_alarm\_subsecond\_get

The description of rtc\_alarm\_subsecond\_get is shown as below:

**Table 3-638. Function rtc\_alarm\_subsecond\_get**

<b>Function name</b>	rtc_alarm_subsecond_get
<b>Function prototype</b>	uint32_t rtc_alarm_subsecond_get(uint8_t rtc_alarm);
<b>Function descriptions</b>	get RTC alarm subsecond
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>rtc_alarm</b>	select alarm0 or alarm1
<i>RTC_ALARM0</i>	alarm0
<i>RTC_ALARM1</i>	alarm1
Output parameter{out}	
-	-
Return value	
<b>uint32_t</b>	RTC alarm subsecond value(0x0-0x3FFF)



Example:

```
/*get RTC alarm0 subsecond*/
```

```
uint32_t subsecond = rtc_alarm_subsecond_get(RTC_ALARM0);
```

### rtc\_alarm\_enable

The description of rtc\_alarm\_enable is shown as below:

**Table 3-639. Function rtc\_alarm\_enable**

<b>Function name</b>	rtc_alarm_enable
<b>Function prototype</b>	void rtc_alarm_enable(uint8_t rtc_alarm);
<b>Function descriptions</b>	enable RTC alarm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_alarm</b>	select alarm0 or alarm1
<i>RTC_ALARM0</i>	alarm0
<i>RTC_ALARM1</i>	alarm1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*enable RTC alarm0*/
```

```
rtc_alarm_enable(RTC_ALARM0);
```

### rtc\_alarm\_disable

The description of rtc\_alarm\_disable is shown as below:

**Table 3-640. Function rtc\_alarm\_disable**

<b>Function name</b>	rtc_alarm_disable
<b>Function prototype</b>	ErrStatus rtc_alarm_disable(uint8_t rtc_alarm);
<b>Function descriptions</b>	disable RTC alarm

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_alarm</b>	select alarm0 or alarm1
<i>RTC_ALARM0</i>	alarm0
<i>RTC_ALARM1</i>	alarm1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/*disable RTC alarm0*/
```

```
ErrStatus error_status = rtc_alarm_disable(RTC_ALARM0);
```

### rtc\_timestamp\_enable

The description of rtc\_timestamp\_enable is shown as below:

**Table 3-641. Function rtc\_timestamp\_enable**

<b>Function name</b>	rtc_timestamp_enable
<b>Function prototype</b>	void rtc_timestamp_enable(uint32_t edge);
<b>Function descriptions</b>	enable RTC time-stamp
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>edge</b>	specify which edge to detect of time-stamp
<i>RTC_TIMESTAMP_RISING_EDGE</i>	rising edge is valid event edge for timestamp event
<i>RTC_TIMESTAMP_FALLING_EDGE</i>	falling edge is valid event edge for timestamp event
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/*enable RTC time-stamp*/
```

```
rtc_timestamp_enable (RTC_TIMESTAMP_RISING_EDGE);
```

### rtc\_timestamp\_disable

The description of rtc\_timestamp\_disable is shown as below:

**Table 3-642. Function rtc\_timestamp\_disable**

Function name	rtc_timestamp_disable
Function prototype	void rtc_timestamp_disable(void);
Function descriptions	disable RTC time-stamp
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*disable RTC time-stamp*/
```

```
rtc_timestamp_disable ();
```

### rtc\_timestamp\_get

The description of rtc\_timestamp\_get is shown as below:

**Table 3-643. Function rtc\_timestamp\_get**

Function name	rtc_timestamp_get
Function prototype	void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);
Function descriptions	get RTC timestamp time and date

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>rtc_timestamp</b>	the structure members can refer to members of the structure <a href="#">Table 3-627</a> . <a href="#">Struct rtc_tamper_struct</a>
<b>Return value</b>	
-	-

Example:

```
/* get RTC timestamp time and date */
rtc_timestamp_struct rtc_timestamp;
rtc_timestamp_get(& rtc_timestamp);
```

### rtc\_timestamp\_subsecond\_get

The description of rtc\_timestamp\_subsecond\_get is shown as below:

**Table 3-644. Function rtc\_timestamp\_subsecond\_get**

<b>Function name</b>	rtc_timestamp_subsecond_get
<b>Function prototype</b>	uint32_t rtc_timestamp_subsecond_get(void);
<b>Function descriptions</b>	get RTC time-stamp subsecond
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	RTC time-stamp subsecond value

Example:

```
/* get RTC time-stamp subsecond */
```

```
uint32_t subsecond = rtc_timestamp_subsecond_get();
```

### rtc\_timestamp\_pin\_map

The description of rtc\_timestamp\_pin\_map is shown as below:

**Table 3-645. Function rtc\_timestamp\_pin\_map**

<b>Function name</b>	rtc_timestamp_pin_map
<b>Function prototype</b>	void rtc_timestamp_pin_map(uint32_t rtc_af);
<b>Function descriptions</b>	RTC time-stamp mapping
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_af</b>	Timestamp input mapping selection
<i>RTC_AF0_TIMESTAMP</i> <i>P</i>	RTC_AF0 use for timestamp
<i>RTC_AF1_TIMESTAMP</i> <i>P</i>	RTC_AF1 use for timestamp
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* RTC time-stamp mapping */
```

```
rtc_timestamp_pin_map (RTC_AF0_TIMESTAMP);
```

### rtc\_tamper\_enable

The description of rtc\_tamper\_enable is shown as below:

**Table 3-646. Function rtc\_timestamp\_enable**

<b>Function name</b>	rtc_tamper_enable
<b>Function prototype</b>	void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);
<b>Function descriptions</b>	enable RTC tamper
<b>Precondition</b>	-

The called functions	-
Input parameter{in}	
rtc_tamper	the structure members can refer to members of the structure <a href="#">Table 3-627</a> . <a href="#">Struct rtc_tamper_struct</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable RTC tamper */
rtc_tamper_struct rtc_tamper
rtc_tamper_enable(& rtc_tamper);
```

### rtc\_tamper\_disable

The description of rtc\_tamper\_disable is shown as below:

**Table 3-647. Function rtc\_tamper\_disable**

Function name	rtc_tamper_disable
Function prototype	void rtc_tamper_disable(uint32_t source);
Function descriptions	disable RTC tamper
Precondition	-
The called functions	-
Input parameter{in}	
source	specify which tamper source to be disabled
RTC_TAMPER0	RTC tamper0
RTC_TAMPER1	RTC tamper1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RTC tamper */
```

```
rtc_tamper_disable(RTC_TAMPER0);
```

### rtc\_tamper0\_pin\_map

The description of rtc\_tamper0\_pin\_map is shown as below:

**Table 3-648. Function rtc\_tamper0\_pin\_map**

<b>Function name</b>	rtc_tamper0_pin_map
<b>Function prototype</b>	void rtc_tamper0_pin_map (uint32_t rtc_af);
<b>Function descriptions</b>	RTC tamper0 mapping
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_af</b>	Tamper0 fuction input mapping selection
<i>RTC_AF0_TAMPER0</i>	RTC_AF0 use for tamper0
<i>RTC_AF1_TAMPER0</i>	RTC_AF1 use for tamper0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* RTC tamper0 mapping */
```

```
rtc_tamper0_pin_map (RTC_AF0_TAMPER0);
```

### rtc\_interrupt\_enable

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

**Table 3-649. Function rtc\_interrupt\_enable**

<b>Function name</b>	rtc_interrupt_enable
<b>Function prototype</b>	void rtc_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable specified RTC interrupt
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which interrupt source to be enabled
<i>RTC_INT_TIMESTAMP</i>	timestamp interrupt
<i>RTC_INT_ALARM0</i>	Alarm0 interrupt
<i>RTC_INT_ALARM1</i>	Alarm1 interrupt
<i>RTC_INT_TAMP</i>	tamp interrupt
<i>RTC_INT_WAKEUP</i>	wakeup timer interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable specified RTC interrupt*/
```

```
rtc_interrupt_enable(RTC_INT_TAMP);
```

### rtc\_interrupt\_disable

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

**Table 3-650. Function rtc\_interrupt\_disable**

<b>Function name</b>	rtc_interrupt_disable
<b>Function prototype</b>	void rtc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disble specified RTC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which RTC interrupt to disable
<i>RTC_INT_TIMESTAMP</i>	second interrupt
<i>RTC_INT_ALARM0</i>	alarm0 interrupt
<i>RTC_INT_ALARM1</i>	alarm1 interrupt



<i>RTC_INT_TAMP</i>	tamp interrupt
<i>RTC_INT_WAKEUP</i>	wakeup timer interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable specified RTC interrupt */
```

```
rtc_interrupt_disable(RTC_INT_TAMP);
```

### rtc\_flag\_get

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

**Table 3-651. Function rtc\_flag\_get**

<b>Function name</b>	rtc_flag_get
<b>Function prototype</b>	FlagStatus rtc_flag_get(uint32_t flag);
<b>Function descriptions</b>	check specified flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag to check
<i>RTC_STAT_SCP</i>	smooth calibration pending flag
<i>RTC_FLAG_TP1</i>	RTC tamper 1 detected flag
<i>RTC_FLAG_TP0</i>	RTC tamper 0 detected flag
<i>RTC_FLAG_TSOVR</i>	time-stamp overflow flag
<i>RTC_FLAG_TS</i>	time-stamp flag
<i>RTC_FLAG_ALARM0</i>	alarm0 occurs flag
<i>RTC_FLAG_ALARM1</i>	Alarm1 occurs flag
<i>RTC_FLAG_WT</i>	wakeup timer occurs flag
<i>RTC_FLAG_INIT</i>	initialization state flag

<i>RTC_FLAG_RSYN</i>	register synchronization flag
<i>RTC_FLAG_YCM</i>	year configuration mark status flag
<i>RTC_FLAG_SOP</i>	alarm0 configuration can be write flag
<i>RTC_FLAG_ALRM0W</i>	alarm0 writen available flag
<i>RTC_FLAG_ALRM1W</i>	alarm1 writen available flag
<i>RTC_FLAG_WTW</i>	wakeup timer can be write flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
FlagStatus	SET or RESET

Example:

```
/* check time-stamp event flag */
```

```
FlagStatus = rtc_flag_get(RTC_FLAG_TIMESTAMP);
```

### rtc\_flag\_clear

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

**Table 3-652. Function rtc\_flag\_clear**

<b>Function name</b>	rtc_flag_clear
<b>Function prototype</b>	void rtc_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear specified flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag to clear
<i>RTC_FLAG_TP1</i>	RTC tamper 1 detected flag
<i>RTC_FLAG_TP0</i>	RTC tamper 0 detected flag
<i>RTC_FLAG_TSOVR</i>	time-stamp overflow flag
<i>RTC_FLAG_TS</i>	time-stamp flag
<i>RTC_FLAG_WT</i>	wakeup timer occurs flag

<i>RTC_FLAG_ALARM0</i>	alarm0 occurs flag
<i>RTC_FLAG_ALARM1</i>	alarm1 occurs flag
<i>RTC_FLAG_RSYN</i>	register synchronization flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* cleartime-stamp event flag */
```

```
rtc_flag_clear (RTC_FLAG_TIMESTAMP);
```

### rtc\_alter\_output\_config

The description of rtc\_alter\_output\_config is shown as below:

**Table 3-653. Function rtc\_alter\_output\_config**

<b>Function name</b>	rtc_alter_output_config
<b>Function prototype</b>	void rtc_alter_output_config(uint32_t source, uint32_t mode);
<b>Function descriptions</b>	configure rtc alternate output source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	specify signal to output
<i>RTC_ALARM0_HIGH</i>	when the alarm0 flag is set, the output pin is high
<i>RTC_ALARM0_LOW</i>	when the alarm0 flag is set, the output pin is low
<i>RTC_ALARM1_HIGH</i>	when the alarm1 flag is set, the output pin is high
<i>RTC_ALARM1_LOW</i>	when the alarm1 flag is set, the output pin is low
<i>RTC_WAKEUP_HIGH</i>	when the wakeup flag is set, the output pin is high
<i>RTC_WAKEUP_LOW</i>	when the wakeup flag is set, the output pin is low
<b>Input parameter{in}</b>	
<b>mode</b>	specify the output pin (PC13) mode when output alarm signal

<i>RTC_ALARM_OUTPU</i> <i>T_OD</i>	open drain mode
<i>RTC_ALARM_OUTPU</i> <i>T_PP</i>	push pull mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure rtc alternate output source */
```

```
rtc_alter_output_config(RTC_ALARM0_LOW, RTC_ALARM_OUTPUT_PP);
```

### rtc\_calibration\_output\_config

The description of rtc\_calibration\_output\_config is shown as below:

**Table 3-654. Function rtc\_calibration\_output\_config**

<b>Function name</b>	rtc_calibration_output_config
<b>Function prototype</b>	void rtc_calibration_output_config(uint32_t source)
<b>Function descriptions</b>	configure rtc calibration output source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	specify signal to output
<i>RTC_CALIBRATION_5</i> <i>12HZ</i>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 512Hz signal
<i>RTC_CALIBRATION_1</i> <i>HZ</i>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 1Hz signal
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure rtc calibration output source*/
```

```
rtc_calibration_output_config (RTC_CALIBRATION_1HZ);
```

### rtc\_hour\_adjust

The description of rtc\_hour\_adjust is shown as below:

**Table 3-655. Function rtc\_hour\_adjust**

<b>Function name</b>	rtc_hour_adjust
<b>Function prototype</b>	void rtc_hour_adjust(uint32_t operation);
<b>Function descriptions</b>	adjust the daylight saving time by adding or subtracting one hour from the current time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>operation</b>	hour adjustment operation
<i>RTC_CTL_A1H</i>	add one hour
<i>RTC_CTL_S1H</i>	subtract one hour
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* adjust the daylight saving time by adding one hour from the current time */
```

```
rtc_hour_adjust(RTC_CTL_A1H);
```

### rtc\_second\_adjust

The description of rtc\_second\_adjust is shown as below:

**Table 3-656. Function rtc\_second\_adjust**

<b>Function name</b>	rtc_second_adjust
<b>Function prototype</b>	ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);
<b>Function descriptions</b>	adjust RTC second or subsecond value of current time
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>add</b>	add 1s to current time or not
<i>RTC_SHIFT_ADD1S_R ESET</i>	no effect
<i>RTC_SHIFT_ADD1S_S ET</i>	add 1s to current time
<b>Input parameter{in}</b>	
<b>minus</b>	number of subsecond to minus from current time(0x0 - 0x7FFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* adjust RTC second or subsecond value of current time */
```

```
ErrStatus error_status = rtc_second_adjust(RTC_SHIFT_ADD1S_SET, 0);
```

### rtc\_bypass\_shadow\_enable

The description of rtc\_bypass\_shadow\_enable is shown as below:

**Table 3-657. Function rtc\_bypass\_shadow\_enable**

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

-	-
---	---

Example:

```
/* enable RTC bypass shadow registers function*/
```

```
rtc_bypass_shadow_enable();
```

### rtc\_bypass\_shadow\_disable

The description of rtc\_bypass\_shadow\_disable shown as below:

**Table 3-658. Function rtc\_bypass\_shadow\_disable**

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

Example:

```
/* disable RTC bypass shadow registers function*/
```

```
rtc_bypass_shadow_disable ();
```

### rtc\_refclock\_detection\_enable

The description of rtc\_refclock\_detection\_enable shown as below:

**Table 3-659. Function rtc\_refclock\_detection\_enable**

<b>Function name</b>	rtc_refclock_detection_enable
<b>Function prototype</b>	ErrStatus rtc_refclock_detection_enable(void);
<b>Function descriptions</b>	enable RTC reference clock detection function
<b>Precondition</b>	-

<b>The called functions</b>	rtc_init_mode_enter/rtc_init_mode_exit
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* enable RTC reference clock detection function*/
```

```
ErrStatus error_status = rtc_refclock_detection_enable();
```

### rtc\_refclock\_detection\_disable

The description of rtc\_refclock\_detection\_disable shown as below:

**Table 3-660. Function rtc\_refclock\_detection\_disable**

<b>Function name</b>	rtc_refclock_detection_disable
<b>Function prototype</b>	ErrStatus rtc_refclock_detection_disable(void);
<b>Function descriptions</b>	disable RTC reference clock detection function
<b>Precondition</b>	-
<b>The called functions</b>	rtc_init_mode_enter/rtc_init_mode_exit
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* disable RTC reference clock detection function*/
```

```
ErrStatus error_status = rtc_refclock_detection_disable ();
```



## rtc\_wakeup\_enable

The description of rtc\_wakeup\_enable shown as below:

**Table 3-661. Function rtc\_wakeup\_enable**

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

Example:

```
/* enable RTC auto wakeup function */
```

```
rtc_wakeup_enable ();
```

## rtc\_wakeup\_disable

The description of rtc\_wakeup\_disable shown as below:

**Table 3-662. Function rtc\_wakeup\_disable**

<b>Function name</b>	rtc_wakeup_disable
<b>Function prototype</b>	ErrStatus rtc_wakeup_disable(void);
<b>Function descriptions</b>	disable RTC auto wakeup function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* disable RTC auto wakeup function */
```

```
rtc_wakeup_disable ();
```

### rtc\_wakeup\_clock\_set

The description of rtc\_wakeup\_clock\_set shown as below:

**Table 3-663. Function rtc\_wakeup\_clock\_set**

Function name	rtc_wakeup_clock_set
Function prototype	ErrStatus rtc_wakeup_clock_set(uint8_t wakeup_clock);
Function descriptions	set RTC auto wakeup timer clock
Precondition	-
The called functions	-
Input parameter{in}	
wakeup_clock	Auto-wakeup timer clock selection
WAKEUP_RTCK_DIV 16	RTC auto wakeup timer clock is RTC clock divided by 16
WAKEUP_RTCK_DIV 8	RTC auto wakeup timer clock is RTC clock divided by 8
WAKEUP_RTCK_DIV 4	RTC auto wakeup timer clock is RTC clock divided by 4
WAKEUP_RTCK_DIV 2	RTC auto wakeup timer clock is RTC clock divided by 2
WAKEUP_CKSPRE	RTC auto wakeup timer clock is ckspre
WAKEUP_CKSPRE_2 EXP16	RTC auto wakeup timer clock is ckspre and wakeup timer add 2exp16
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* set RTC auto wakeup timer clock */
```

```
ErrStatus error_status = rtc_wakeup_clock_set (WAKEUP_RTCK_DIV16);
```

### rtc\_wakeup\_timer\_set

The description of rtc\_wakeup\_timer\_set shown as below:

**Table 3-664. Function rtc\_wakeup\_timer\_set**

<b>Function name</b>	rtc_wakeup_timer_set
<b>Function prototype</b>	ErrStatus rtc_wakeup_timer_set(uint16_t wakeup_timer);
<b>Function descriptions</b>	set wakeup timer value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
wakeup_timer	Auto_wakeup timer reloads value(0x0000-0xFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* set wakeup timer value */
```

```
ErrStatus error_status = rtc_wakeup_timer_set (0XFFEE);
```

### rtc\_wakeup\_timer\_get

The description of rtc\_wakeup\_timer\_get shown as below:

**Table 3-665. Function rtc\_wakeup\_timer\_get**

<b>Function name</b>	rtc_wakeup_timer_get
<b>Function prototype</b>	uint16_t rtc_wakeup_timer_get(void);
<b>Function descriptions</b>	get wakeup timer value
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	0-0XFFFF

Example:

```
/* get wakeup timer value */
```

```
rtc_wakeup_timer_get ();
```

### rtc\_smooth\_calibration\_config

The description of rtc\_smooth\_calibration\_config shown as below:

**Table 3-666. Function rtc\_smooth\_calibration\_config**

Function name	rtc_smooth_calibration_config
Function prototype	ErrStatus rtc_smooth_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);
Function descriptions	configure RTC smooth calibration
Precondition	-
The called functions	-
Input parameter{in}	
window	select calibration window
RTC_CALIBRATION_WINDOW_32S	2exp20 RTCCLK cycles, 32s if RTCCLK = 32768 Hz
RTC_CALIBRATION_WINDOW_16S	2exp19 RTCCLK cycles, 16s if RTCCLK = 32768 Hz
RTC_CALIBRATION_WINDOW_8S	2exp18 RTCCLK cycles, 8s if RTCCLK = 32768 Hz
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* configure RTC smooth calibration */
```

```
ErrStatus error_status;
```

```
error_status = rtc_smooth_calibration_config(RTC_CALIBRATION_WINDOW_32S,
RTC_CALIBRATION_PLUS_SET);
```

### rtc\_coarse\_calibration\_enable

The description of rtc\_coarse\_calibration\_enable shown as below:

**Table 3-667. Function rtc\_coarse\_calibration\_enable**

<b>Function name</b>	rtc_coarse_calibration_enable
<b>Function prototype</b>	ErrStatus rtc_coarse_calibration_enable(void);
<b>Function descriptions</b>	enable RTC coarse calibration
<b>Precondition</b>	-
<b>The called functions</b>	rtc_init_mode_enter/rtc_init_mode_exit
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* enable RTC coarse calibration */
```

```
rtc_coarse_calibration_enable ();
```

### rtc\_coarse\_calibration\_disable

The description of rtc\_coarse\_calibration\_disable shown as below:

**Table 3-668. Function rtc\_coarse\_calibration\_disable**

<b>Function name</b>	rtc_coarse_calibration_disable
<b>Function prototype</b>	ErrStatus rtc_coarse_calibration_disable(void);
<b>Function descriptions</b>	disable RTC coarse calibration
<b>Precondition</b>	-

<b>The called functions</b>	rtc_init_mode_enter/rtc_init_mode_exit
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* rtc_coarse_calibration_disable */
```

```
ErrStatus error_status = rtc_coarse_calibration_disable ();
```

### rtc\_coarse\_calibration\_config

The description of rtc\_coarse\_calibration\_config shown as below:

**Table 3-669. Function rtc\_coarse\_calibration\_config**

<b>Function name</b>	rtc_coarse_calibration_config
<b>Function prototype</b>	ErrStatus rtc_coarse_calibration_config(uint8_t direction, uint8_t step);
<b>Function descriptions</b>	config coarse calibration direction and step
<b>Precondition</b>	-
<b>The called functions</b>	rtc_init_mode_enter/rtc_init_mode_exit
<b>Input parameter{in}</b>	
<b>direction</b>	coarse calibration direction
<i>CALIB_INCREASE</i>	Increase calendar update frequency
<i>CALIB_DECREASE</i>	Decrease calendar update frequency
<b>Input parameter{in}</b>	
<b>step</b>	Coarse calibration direction
<i>0x00-0x1F</i>	COSD=0: 0x00: +0PPM 0x01: +4PPM(approximate value) 0x02: +8PPM (approximate value) ..... 0x1F: +126PPM (approximate value)

	COSD=1: 0x00: -0PPM 0x01: -2PPM(approximate value) 0x02: -4PPM (approximate value) ..... 0x1F: -63PPM (approximate value)
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* config coarse calibration direction and step */
```

```
ErrStatus error_status = rtc_coarse_calibration_config (INCREASE, 0x01);
```

## 3.23. SDIO

The secure digital input/output interface (SDIO) defines the SD/SD I/O /MMC CE-ATA card host interface, which provides command/data transfer between the APB2 system bus and SD memory cards, SD I/O cards, Multimedia Card (MMC), and CE-ATA devices. The SDIO registers are listed in chapter [3.23.1](#), the SDIO firmware functions are introduced in chapter [3.23.2](#).

### 3.23.1. Descriptions of Peripheral registers

SDIO registers are listed in the table shown as below:

**Table 3-670. SDIO Registers**

Registers	Descriptions
SDIO_PWRCTL	Power control register
SDIO_CLKCTL	Clock control register
SDIO_CMDAGMT	Command argument register
SDIO_CMDCTL	Command control register
SDIO_RSPCMDIDX	Command index response register
SDIO_RESPx x=0..3	Response register
SDIO_DATATO	Data timeout register

Registers	Descriptions
SDIO_DATALEN	Data length register
SDIO_DATACTL	Data control register
SDIO_DATACNT	Data counter register
SDIO_STAT	Status register
SDIO_INTC	Interrupt clear register
SDIO_INTEN	Interrupt enable register
SDIO_FIFOCNT	FIFO counter register
SDIO_FIFO	FIFO data register

### 3.23.2. Descriptions of Peripheral functions

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

**Table 3-671. SDIO firmware function**

Function name	Function description
sdio_deinit	deinitialize the SDIO
sdio_clock_config	configure the SDIO clock
sdio_hardware_clock_enable	enable hardware clock control
sdio_hardware_clock_disable	disable hardware clock control
sdio_bus_mode_set	set different SDIO card bus mode
sdio_power_state_set	set the SDIO power state
sdio_power_state_get	get the SDIO power state
sdio_clock_enable	enable SDIO_CLK clock output
sdio_clock_disable	disable SDIO_CLK clock output
sdio_command_response_config	configure the command and response
sdio_wait_type_set	set the command state machine wait type
sdio_csm_enable	enable the CSM(command state machine)
sdio_csm_disable	disable the CSM(command state machine)
sdio_command_index_get	get the last response command index
sdio_response_get	get the response for the last received command



Function name	Function description
sdio_data_config	configure the data timeout, data length and data block size
sdio_data_transfer_config	configure the data transfer mode and direction
sdio_dsm_enable	enable the DSM(data state machine) for data transfer
sdio_dsm_disable	disable the DSM(data state machine)
sdio_data_write	write data(one word) to the transmit FIFO
sdio_data_read	read data(one word) from the receive FIFO
sdio_data_counter_get	get the number of remaining data bytes to be transferred to card
sdio_fifo_counter_get	get the number of words remaining to be written or read from FIFO
sdio_dma_enable	enable the DMA request for SDIO
sdio_dma_disable	disable the DMA request for SDIO
sdio_flag_get	get the flags state of SDIO
sdio_flag_clear	clear the pending flags of SDIO
sdio_interrupt_enable	enable the SDIO interrupt
sdio_interrupt_disable	disable the SDIO interrupt
sdio_interrupt_flag_get	get the interrupt flags state of SDIO
sdio_interrupt_flag_clear	clear the interrupt pending flags of SDIO
sdio_readwait_enable	enable the read wait mode(SD I/O only)
sdio_readwait_disable	disable the read wait mode(SD I/O only)
sdio_stop_readwait_enable	enable the function that stop the read wait process(SD I/O only)
sdio_stop_readwait_disable	disable the function that stop the read wait process(SD I/O only)
sdio_readwait_type_set	set the read wait type(SD I/O only)
sdio_operation_enable	enable the SD I/O mode specific operation(SD I/O only)
sdio_operation_disable	disable the SD I/O mode specific operation(SD I/O only)
sdio_suspend_enable	enable the SD I/O suspend operation(SD I/O only)
sdio_suspend_disable	disable the SD I/O suspend operation(SD I/O only)

Function name	Function description
sdio_ceata_command_enable	enable the CE-ATA command(CE-ATA only)
sdio_ceata_command_disable	disable the CE-ATA command(CE-ATA only)
sdio_ceata_interrupt_enable	enable the CE-ATA interrupt(CE-ATA only)
sdio_ceata_interrupt_disable	disable the CE-ATA interrupt(CE-ATA only)
sdio_ceata_command_completion_enable	enable the CE-ATA command completion signal(CE-ATA only)
sdio_ceata_command_completion_disable	disable the CE-ATA command completion signal(CE-ATA only)
sdio_clock_config	configure the SDIO clock

### sdio\_deinit

The description of sdio\_deinit is shown as below:

**Table 3-672. Function sdio\_deinit**

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

Example:

```
/* deinitialize the SDIO */
sdio_deinit();
```

### sdio\_clock\_config

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

Table 3-673. Function `sdio_clock_config`

<b>Function name</b>	<code>sdio_clock_config</code>
<b>Function prototype</b>	<code>void sdio_clock_config(uint32_t clock_edge, uint32_t clock_bypass, uint32_t clock_powersave, uint16_t clock_division);</code>
<b>Function descriptions</b>	configure the SDIO clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>clock_edge</b>	SDIO_CLK clock edge
<code>SDIO_SDIOLCKEDGE_RISING</code>	select the rising edge of the SDIOCLK to generate SDIO_CLK
<code>SDIO_SDIOLCKEDGE_FALLING</code>	select the falling edge of the SDIOCLK to generate SDIO_CLK
<b>Input parameter{in}</b>	
<b>clock_bypass</b>	clock bypass
<code>SDIO_CLOCKBYPASS_ENABLE</code>	clock bypass
<code>SDIO_CLOCKBYPASS_DISABLE</code>	no bypass
<b>Input parameter{in}</b>	
<b>clock_powersave</b>	SDIO_CLK clock dynamic switch on/off for power saving
<code>SDIO_CLOCKPW RSA_VE_ENABLE</code>	SDIO_CLK closed when bus is idle
<code>SDIO_CLOCKPW RSA_VE_DISABLE</code>	SDIO_CLK clock is always on
<b>Input parameter{in}</b>	
<b>clock_division</b>	clock division, less than 512
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the SDIO clock */
```

```
sdio_clock_config(SDIO_SDIOLCKEDGE_RISING, SDIO_CLOCKBYPASS_DISABLE,
SDIO_CLOCKPWRSAVE_DISABLE, SD_CLK_DIV_TRANS);
```

### sdio\_hardware\_clock\_enable

The description of sdio\_hardware\_clock\_enable is shown as below:

**Table 3-674. Function sdio\_hardware\_clock\_enable**

<b>Function name</b>	sdio_hardware_clock_enable
<b>Function prototype</b>	void sdio_hardware_clock_enable(void);
<b>Function descriptions</b>	enable hardware clock control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable hardware clock control */
```

```
sdio_hardware_clock_enable();
```

### sdio\_hardware\_clock\_disable

The description of sdio\_hardware\_clock\_disable is shown as below:

**Table 3-675. Function sdio\_hardware\_clock\_disable**

<b>Function name</b>	sdio_hardware_clock_disable
<b>Function prototype</b>	void sdio_hardware_clock_disable(void);
<b>Function descriptions</b>	disable hardware clock control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

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

Example:

```
/* disable hardware clock control */
```

```
sdio_hardware_clock_disable();
```

### sdio\_bus\_mode\_set

The description of sdio\_bus\_mode\_set is shown as below:

**Table 3-676. Function sdio\_bus\_mode\_set**

<b>Function name</b>	sdio_bus_mode_set
<b>Function prototype</b>	void sdio_bus_mode_set(uint32_t bus_mode);
<b>Function descriptions</b>	set different SDIO card bus mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>bus_mode</b>	SDIO card bus mode
<i>SDIO_BUSMODE_1BIT</i>	1-bit SDIO card bus mode
<i>SDIO_BUSMODE_4BIT</i>	4-bit SDIO card bus mode
<i>SDIO_BUSMODE_8BIT</i>	8-bit SDIO card bus mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set SDIO bus mode */
```

```
sdio_bus_mode_set(SDIO_BUSMODE_1BIT);
```

### sdio\_power\_state\_set

The description of sdio\_power\_state\_set is shown as below:

**Table 3-677. Function sdio\_power\_state\_set**

<b>Function name</b>	sdio_power_state_set
<b>Function prototype</b>	void sdio_power_state_set(uint32_t power_state);
<b>Function descriptions</b>	set the SDIO power state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>power_state</b>	SDIO power state
SDIO_POWER_ON	SDIO power on
SDIO_POWER_OFF	SDIO power off
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set SDIO power state */
```

```
sdio_power_state_set(SDIO_POWER_ON);
```

### sdio\_power\_state\_get

The description of sdio\_power\_state\_get is shown as below:

**Table 3-678. Function sdio\_power\_state\_get**

<b>Function name</b>	sdio_power_state_get
<b>Function prototype</b>	uint32_t sdio_power_state_get(void);
<b>Function descriptions</b>	get the SDIO power state
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	SDIO_POWER_ON / SDIO_POWER_OFF

Example:

```

/* get the SDIO power state */

uint32_t sdio_power_value;

sdio_power_value = sdio_power_state_get();

```

### sdio\_clock\_enable

The description of sdio\_clock\_enable is shown as below:

**Table 3-679. Function sdio\_clock\_enable**

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

Example:

```

/* enable SDIO_CLK clock output */

sdio_clock_enable();

```

## sdio\_clock\_disable

The description of sdio\_clock\_disable is shown as below:

**Table 3-680. Function sdio\_clock\_disable**

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

Example:

```
/* disable SDIO_CLK clock output */
```

```
sdio_clock_disable();
```

## sdio\_command\_response\_config

The description of sdio\_command\_response\_config is shown as below:

**Table 3-681. Function sdio\_command\_response\_config**

<b>Function name</b>	sdio_command_response_config
<b>Function prototype</b>	void sdio_command_response_config(uint32_t cmd_index, uint32_t cmd_argument, uint32_t response_type);
<b>Function descriptions</b>	configure the command and response
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>cmd_index</b>	command index, refer to the related specifications
<b>Input parameter{in}</b>	



<b>cmd_argument</b>	command argument, refer to the related specifications
<b>Input parameter{in}</b>	
<b>response_type</b>	response type
<i>SDIO_RESPONSETYPE_NO</i>	no response
<i>SDIO_RESPONSETYPE_SHORT</i>	short response
<i>SDIO_RESPONSETYPE_LONG</i>	long response
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* CMD2(SD_CMD_ALL_SEND_CID) command response config*/
```

```
sdio_command_response_config(SD_CMD_ALL_SEND_CID, (uint32_t)0x0, SDIO_RESPONSETYPE_LONG);
```

### sdio\_wait\_type\_set

The description of sdio\_wait\_type\_set is shown as below:

**Table 3-682. Function sdio\_wait\_type\_set**

<b>Function name</b>	sdio_wait_type_set
<b>Function prototype</b>	void sdio_wait_type_set(uint32_t wait_type);
<b>Function descriptions</b>	set the command state machine wait type
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>wait_type</b>	wait type
<i>SDIO_WAITTYPE_NO</i>	not wait interrupt
<i>SDIO_WAITTYPE_INTERRUPT</i>	wait interrupt

SDIO_WAITTYPE_DATAEND	wait the end of data transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the command state machine wait type */
sdio_wait_type_set(SDIO_WAITTYPE_NO);
```

### sdio\_csm\_enable

The description of sdio\_csm\_enable is shown as below:

**Table 3-683. Function sdio\_csm\_enable**

Function name	sdio_csm_enable
Function prototype	void sdio_csm_enable(void);
Function descriptions	enable the CSM(command state machine)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CSM(command state machine) */
sdio_csm_enable();
```

### sdio\_csm\_disable

The description of sdio\_csm\_disable is shown as below:

Table 3-684. Function `sdio_csm_disable`

Function name	<code>sdio_csm_disable</code>
Function prototype	<code>void sdio_csm_disable(void);</code>
Function descriptions	disable the CSM(command state machine)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CSM(command state machine) */
sdio_csm_disable();
```

### `sdio_command_index_get`

The description of `sdio_command_index_get` is shown as below:

Table 3-685. Function `sdio_command_index_get`

Function name	<code>sdio_command_index_get</code>
Function prototype	<code>uint8_t sdio_command_index_get(void);</code>
Function descriptions	get the last response command index
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
<code>uint8_t</code>	last response command index

Example:

```
/* get SDIO command index */
uint8_t sdio_commond_value;

sdio_commond_value = sdio_command_index_get();
```

### sdio\_response\_get

The description of sdio\_response\_get is shown as below:

**Table 3-686. Function sdio\_response\_get**

Function name	sdio_response_get
Function prototype	uint32_t sdio_response_get(uint32_t responsex);
Function descriptions	get the response for the last received command
Precondition	-
The called functions	-
Input parameter{in}	
responsex	SDIO response
SDIO_RESPONSE0	card response[31:0]/card response[127:96]
SDIO_RESPONSE1	card response[95:64]
SDIO_RESPONSE2	card response[63:32]
SDIO_RESPONSE3	card response[31:1], plus bit 0
Output parameter{out}	
-	-
Return value	
uint32_t	response for the last received command

Example:

```
/* store the CID0 numbers */
uint32_t sdio_cid[0];

sdio_cid[0] = sdio_response_get(SDIO_RESPONSE0);
```

### sdio\_data\_config

The description of sdio\_data\_config is shown as below:

Table 3-687. Function `sdio_data_config`

<b>Function name</b>	<code>sdio_data_config</code>
<b>Function prototype</b>	<code>void sdio_data_config(uint32_t data_timeout, uint32_t data_length, uint32_t data_blocksize);</code>
<b>Function descriptions</b>	configure the data timeout, data length and data block size
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data_timeout</b>	data timeout period in card bus clock periods
<b>Input parameter{in}</b>	
<b>data_length</b>	number of data bytes to be transferred
<b>Input parameter{in}</b>	
<b>data_blocksize</b>	size of data block for block transfer
<code>SDIO_DATABLOCKSIZE_1BYTE</code>	block size = 1 byte
<code>SDIO_DATABLOCKSIZE_2BYTES</code>	block size = 2 bytes
<code>SDIO_DATABLOCKSIZE_4BYTES</code>	block size = 4 bytes
<code>SDIO_DATABLOCKSIZE_8BYTES</code>	block size = 8 bytes
<code>SDIO_DATABLOCKSIZE_16BYTES</code>	block size = 16 bytes
<code>SDIO_DATABLOCKSIZE_32BYTES</code>	block size = 32 bytes
<code>SDIO_DATABLOCKSIZE_64BYTES</code>	block size = 64 bytes
<code>SDIO_DATABLOCKSIZE_128BYTES</code>	block size = 128 bytes
<code>SDIO_DATABLOCKSIZE_256BYTES</code>	block size = 256 bytes
<code>SDIO_DATABLOCKSIZE_512BYTES</code>	block size = 512 bytes

<i>SDIO_DATABLOCKSIZE_1024BYTES</i>	block size = 1024 bytes
<i>SDIO_DATABLOCKSIZE_2048BYTES</i>	block size = 2048 bytes
<i>SDIO_DATABLOCKSIZE_4096BYTES</i>	block size = 4096 bytes
<i>SDIO_DATABLOCKSIZE_8192BYTES</i>	block size = 8192 bytes
<i>SDIO_DATABLOCKSIZE_16384BYTES</i>	block size = 16384 bytes
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SDIO data */
```

```
sdio_data_config(0, 0, SDIO_DATABLOCKSIZE_1BYTE);
```

### sdio\_data\_transfer\_config

The description of sdio\_data\_transfer\_config is shown as below:

**Table 3-688. Function sdio\_data\_transfer\_config**

<b>Function name</b>	sdio_data_transfer_config
<b>Function prototype</b>	void sdio_data_transfer_config(uint32_t transfer_mode, uint32_t transfer_direction);
<b>Function descriptions</b>	configure the data transfer mode and direction
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>transfer_mode</b>	mode of data transfer
<i>SDIO_TRANSMODE_BLOCK</i>	block transfer
<i>SDIO_TRANSMODE_STREAM</i>	stream transfer or SDIO multibyte transfer

<i>TREAM</i>	
<b>Input parameter{in}</b>	
<b>transfer_direction</b>	data transfer direction, read or write
<i>SDIO_TRANSDIRECTI ON_TOCARD</i>	write data to card
<i>SDIO_TRANSDIRECTI ON_TOSDIO</i>	read data from card
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SDIO data transmisson */
```

```
sdio_data_transfer_config(SDIO_TRANSDIRECTION_TOSDIO,  
SDIO_TRANSMODE_BLOCK);
```

### sdio\_dsm\_enable

The description of sdio\_dsm\_enable is shown as below:

**Table 3-689. Function sdio\_dsm\_enable**

<b>Function name</b>	sdio_dsm_enable
<b>Function prototype</b>	void sdio_dsm_enable(void);
<b>Function descriptions</b>	enable the DSM(data state machine) for data transfer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the DSM(data state machine) */
```

```
sdio_dsm_enable();
```

### sdio\_dsm\_disable

The description of sdio\_dsm\_disable is shown as below:

**Table 3-690. Function sdio\_dsm\_disable**

<b>Function name</b>	sdio_dsm_disable
<b>Function prototype</b>	void sdio_dsm_disable(void);
<b>Function descriptions</b>	disable the DSM(data state machine)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the DSM(data state machine) */
```

```
sdio_dsm_disable();
```

### sdio\_data\_write

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

**Table 3-691. Function sdio\_data\_write**

<b>Function name</b>	sdio_data_write
<b>Function prototype</b>	void sdio_data_write(uint32_t data);
<b>Function descriptions</b>	write data(one word) to the transmit FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	



<b>data</b>	32-bit data write to card
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* write data(one word) to the transmit FIFO */
```

```
sdio_data_write(0x0000 0001);
```

### sdio\_data\_read

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

**Table 3-692. Function sdio\_data\_read**

<b>Function name</b>	sdio_data_read
<b>Function prototype</b>	uint32_t sdio_data_read(void);
<b>Function descriptions</b>	read data(one word) from the receive FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	received data

Example:

```
/* read data(one word) from the receive FIFO */
```

```
sdio_data_read();
```

### sdio\_data\_counter\_get

The description of sdio\_data\_counter\_get is shown as below:

Table 3-693. Function `sdio_data_counter_get`

Function name	<code>sdio_data_counter_get</code>
Function prototype	<code>uint32_t sdio_data_counter_get(void);</code>
Function descriptions	get the number of remaining data bytes to be transferred to card
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
<code>uint32_t</code>	number of remaining data bytes to be transferred

Example:

```
/* get the number of remaining data bytes to be transferred to card */
```

```
uint32_t sdio_data_value;
```

```
sdio_data_value = sdio_data_counter_get();
```

### `sdio_fifo_counter_get`

The description of `sdio_fifo_counter_get` is shown as below:

Table 3-694. Function `sdio_data_counter_get`

Function name	<code>sdio_fifo_counter_get</code>
Function prototype	<code>uint32_t sdio_fifo_counter_get(void);</code>
Function descriptions	get the number of words remaining to be written or read from FIFO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

uint32_t	remaining number of words
----------	---------------------------

Example:

```
/* get the number of words remaining to be written or read from FIFO */
```

```
uint32_t sdio_fifo_value;
```

```
sdio_fifo_value = sdio_fifo_counter_get();
```

### sdio\_dma\_enable

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

**Table 3-695. Function sdio\_dma\_enable**

<b>Function name</b>	sdio_dma_enable
<b>Function prototype</b>	void sdio_dma_enable(void);
<b>Function descriptions</b>	enable the DMA request for SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the SDIO DMA */
```

```
sdio_dma_enable();
```

### sdio\_dma\_disable

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

**Table 3-696. Function sdio\_dma\_disable**

<b>Function name</b>	sdio_dma_disable
<b>Function prototype</b>	void sdio_dma_disable(void);
<b>Function descriptions</b>	disable the DMA request for SDIO

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

Example:

```
/* disable the SDIO DMA */
```

```
sdio_dma_disable();
```

### sdio\_flag\_get

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

**Table 3-697. Function sdio\_flag\_get**

<b>Function name</b>	sdio_flag_get
<b>Function prototype</b>	FlagStatus sdio_flag_get(uint32_t flag);
<b>Function descriptions</b>	get the flags state of SDIO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flags state of SDIO
<i>SDIO_FLAG_CCR CER</i> <i>R</i>	command response received (CRC check failed) flag
<i>SDIO_FLAG_DTCRCE</i> <i>RR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_FLAG_CMDTMO</i> <i>UT</i>	command response timeout flag
<i>SDIO_FLAG_DTTMOU</i> <i>T</i>	data timeout flag
<i>SDIO_FLAG_TXURE</i>	transmit FIFO underrun error occurs flag

<i>SDIO_FLAG_RXORE</i>	received FIFO overrun error occurs flag
<i>SDIO_FLAG_CMDREC V</i>	command response received (CRC check passed) flag
<i>SDIO_FLAG_CMDSEN D</i>	command sent (no response required) flag
<i>SDIO_FLAG_DTEND</i>	data end (data counter, <i>SDIO_DATACNT</i> , is zero) flag
<i>SDIO_FLAG_STBITE</i>	start bit error in the bus flag
<i>SDIO_FLAG_DTBK ND</i>	data block sent/received (CRC check passed) flag
<i>SDIO_FLAG_CMDRUN</i>	command transmission in progress flag
<i>SDIO_FLAG_TXRUN</i>	data transmission in progress flag
<i>SDIO_FLAG_RXRUN</i>	data reception in progress flag
<i>SDIO_FLAG_TFH</i>	transmit FIFO is half empty flag: at least 8 words can be written into the FIFO
<i>SDIO_FLAG_RFH</i>	receive FIFO is half full flag: at least 8 words can be read in the FIFO
<i>SDIO_FLAG_TFF</i>	transmit FIFO is full flag
<i>SDIO_FLAG_RFF</i>	receive FIFO is full flag
<i>SDIO_FLAG_TFE</i>	transmit FIFO is empty flag
<i>SDIO_FLAG_RFE</i>	receive FIFO is empty flag
<i>SDIO_FLAG_TXDTVAL</i>	data is valid in transmit FIFO flag
<i>SDIO_FLAG_RXDTVA L</i>	data is valid in receive FIFO flag
<i>SDIO_FLAG_SDIOINT</i>	SD I/O interrupt received flag
<i>SDIO_FLAG_ATAEND</i>	CE-ATA command completion signal received (only for CMD61) flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the flags state of SDIO */
```

```
FlagStatus flag_value;
```

```
flag_value = sdio_flag_get(SDIO_FLAG_RFH);
```

## sdio\_flag\_clear

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

**Table 3-698. Function sdio\_flag\_clear**

Function name	sdio_flag_clear
Function prototype	void sdio_flag_clear(uint32_t flag);
Function descriptions	clear the pending flags of SDIO
Precondition	-
The called functions	-
Input parameter{in}	
flag	flags state of SDIO
SDIO_FLAG_CCRCE R	command response received (CRC check failed) flag
SDIO_FLAG_DTCRCE RR	data block sent/received (CRC check failed) flag
SDIO_FLAG_CMDTMO UT	command response timeout flag
SDIO_FLAG_DTTMOU T	data timeout flag
SDIO_FLAG_TXURE	transmit FIFO underrun error occurs flag
SDIO_FLAG_RXORE	received FIFO overrun error occurs flag
SDIO_FLAG_CMDREC V	command response received (CRC check passed) flag
SDIO_FLAG_CMDSEN D	command sent (no response required) flag
SDIO_FLAG_DTEND	data end (data counter, SDIO_DATACNT, is zero) flag
SDIO_FLAG_STBITE	start bit error in the bus flag
SDIO_FLAG_DTBKE ND	data block sent/received (CRC check passed) flag
SDIO_FLAG_SDIOINT	SD I/O interrupt received flag
SDIO_FLAG_ATAEND	CE-ATA command completion signal received (only for CMD61) flag

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the pending flags of SDIO */
```

```
sdio_flag_clear(SDIO_FLAG_DTCRCERR);
```

### sdio\_interrupt\_enable

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

**Table 3-699. Function sdio\_interrupt\_enable**

Function name	sdio_interrupt_enable
Function prototype	void sdio_interrupt_enable(uint32_t int_flag);
Function descriptions	enable the SDIO interrupt
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	interrupt flags state of SDIO
SDIO_INT_CCRERR	SDIO CCRERR interrupt
SDIO_INT_DTCRCERR	SDIO DTCRCERR interrupt
SDIO_INT_CMDTMOUT	SDIO CMDTMOUT interrupt
SDIO_INT_DTTMOUT	SDIO DTTMOUT interrupt
SDIO_INT_TXURE	SDIO TXURE interrupt
SDIO_INT_RXORE	SDIO_INT_RXORE
SDIO_INT_CMDRECV	SDIO CMDRECV interrupt
SDIO_INT_CMDSND	SDIO CMDSND interrupt
SDIO_INT_DTEND	SDIO DTEND interrupt
SDIO_INT_STBITE	SDIO STBITE interrupt

<i>SDIO_INT_DTBLKEND</i>	SDIO DTBLKEND interrupt
<i>SDIO_INT_CMDRUN</i>	SDIO CMDRUN interrupt
<i>SDIO_INT_TXRUN</i>	SDIO TXRUN interrupt
<i>SDIO_INT_RXRUN</i>	SDIO RXRUN interrupt
<i>SDIO_INT_TFH</i>	SDIO TFH interrupt
<i>SDIO_INT_RFH</i>	SDIO RFH interrupt
<i>SDIO_INT_TFF</i>	SDIO TFF interrupt
<i>SDIO_INT_RFF</i>	SDIO RFF interrupt
<i>SDIO_INT_TFE</i>	SDIO TFE interrupt
<i>SDIO_INT_RFE</i>	SDIO RFE interrupt
<i>SDIO_INT_TXDTVAL</i>	SDIO TXDTVAL interrupt
<i>SDIO_INT_RXDTVAL</i>	SDIO RXDTVAL interrupt
<i>SDIO_INT_SDIOINT</i>	SDIO SDIOINT interrupt
<i>SDIO_INT_ATAEND</i>	SDIO ATAEND interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the SDIO corresponding interrupts */
```

```
sdio_interrupt_enable(SDIO_INT_CCRCERR | SDIO_INT_DTTMOUT | SDIO_INT_RXORE  
| SDIO_INT_DTEND | SDIO_INT_STBITE);
```

### **sdio\_interrupt\_disable**

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

**Table 3-700. Function sdio\_interrupt\_disable**

<b>Function name</b>	sdio_interrupt_disable
<b>Function prototype</b>	void sdio_interrupt_disable(uint32_t int_flag);
<b>Function descriptions</b>	disable the SDIO interrupt
<b>Precondition</b>	-



The called functions	-
Input parameter{in}	
<b>int_flag</b>	interrupt flags state of SDIO
<i>SDIO_INT_CCRERR</i>	SDIO CCRERR interrupt
<i>SDIO_INT_DTCRCERR</i>	SDIO DTCRCERR interrupt
<i>SDIO_INT_CMDTMOUT</i>	SDIO CMDTMOUT interrupt
<i>SDIO_INT_DTTMOUT</i>	SDIO DTTMOUT interrupt
<i>SDIO_INT_TXURE</i>	SDIO TXURE interrupt
<i>SDIO_INT_RXORE</i>	SDIO_INT_RXORE
<i>SDIO_INT_CMDRECV</i>	SDIO CMDRECV interrupt
<i>SDIO_INT_CMDSEND</i>	SDIO CMDSEND interrupt
<i>SDIO_INT_DTEND</i>	SDIO DTEND interrupt
<i>SDIO_INT_STBITE</i>	SDIO STBITE interrupt
<i>SDIO_INT_DTBLKEND</i>	SDIO DTBLKEND interrupt
<i>SDIO_INT_CMDRUN</i>	SDIO CMDRUN interrupt
<i>SDIO_INT_TXRUN</i>	SDIO TXRUN interrupt
<i>SDIO_INT_RXRUN</i>	SDIO RXRUN interrupt
<i>SDIO_INT_TFH</i>	SDIO TFH interrupt
<i>SDIO_INT_RFH</i>	SDIO RFH interrupt
<i>SDIO_INT_TFF</i>	SDIO TFF interrupt
<i>SDIO_INT_RFF</i>	SDIO RFF interrupt
<i>SDIO_INT_TFE</i>	SDIO TFE interrupt
<i>SDIO_INT_RFE</i>	SDIO RFE interrupt
<i>SDIO_INT_TXDTVAL</i>	SDIO TXDTVAL interrupt
<i>SDIO_INT_RXDTVAL</i>	SDIO RXDTVAL interrupt
<i>SDIO_INT_SDIOINT</i>	SDIO SDIOINT interrupt
<i>SDIO_INT_ATAEND</i>	SDIO ATAEND interrupt

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the SDIO interrupt */
```

```
sdio_interrupt_disable(SDIO_INT_DTCRCERR);
```

### sdio\_interrupt\_flag\_get

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

**Table 3-701. Function sdio\_interrupt\_flag\_get**

Function name	sdio_interrupt_flag_get
Function prototype	FlagStatus sdio_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get the interrupt flags state of SDIO
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	interrupt flags state of SDIO
SDIO_INT_FLAG_CCR CERR	SDIO CCRERR interrupt flag
SDIO_INT_FLAG_DTC RCERR	SDIO DTCRCERR interrupt flag
SDIO_INT_FLAG_CMD TMOUT	SDIO CMDTMOUT interrupt flag
SDIO_INT_FLAG_DTT MOUT	SDIO DTTMOUT interrupt flag
SDIO_INT_FLAG_TXU RE	SDIO TXURE interrupt flag
SDIO_INT_FLAG_RXO RE	SDIO_INT_RXORE flag
SDIO_INT_FLAG_CMD RECV	SDIO CMDRECV interrupt flag

<i>SDIO_INT_FLAG_CMD SEND</i>	SDIO CMDSEND interrupt flag
<i>SDIO_INT_FLAG_DTE ND</i>	SDIO DTEND interrupt flag
<i>SDIO_INT_FLAG_STBI TE</i>	SDIO STBITE interrupt flag
<i>SDIO_INT_FLAG_DTB LKEND</i>	SDIO DTBLKEND interrupt flag
<i>SDIO_INT_FLAG_CMD RUN</i>	SDIO CMDRUN interrupt flag
<i>SDIO_INT_FLAG_TXR UN</i>	SDIO TXRUN interrupt flag
<i>SDIO_INT_FLAG_RXR UN</i>	SDIO RXRUN interrupt flag
<i>SDIO_INT_FLAG_TFH</i>	SDIO TFH interrupt flag
<i>SDIO_INT_FLAG_RFH</i>	SDIO RFH interrupt flag
<i>SDIO_INT_FLAG_TFF</i>	SDIO TFF interrupt flag
<i>SDIO_INT_FLAG_RFF</i>	SDIO RFF interrupt flag
<i>SDIO_INT_FLAG_TFE</i>	SDIO TFE interrupt flag
<i>SDIO_INT_FLAG_RFE</i>	SDIO RFE interrupt flag
<i>SDIO_INT_FLAG_TXD TVAL</i>	SDIO TXDTVAL interrupt flag
<i>SDIO_INT_FLAG_RXD TVAL</i>	SDIO RXDTVAL interrupt flag
<i>SDIO_INT_FLAG_SDI OINT</i>	SDIO SDIOINT interrupt flag
<i>SDIO_INT_FLAG_ATA END</i>	SDIO ATAEND interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the interrupt flags state of SDIO */
```

```
FlagStatus flag_value;
```

```
flag_value = sdio_interrupt_flag_get(SDIO_INT_FLAG_DTEND);
```

### sdio\_interrupt\_flag\_clear

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

**Table 3-702. Function sdio\_interrupt\_flag\_clear**

Function name	sdio_interrupt_flag_clear
Function prototype	void sdio_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear the interrupt pending flags of SDIO
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	interrupt flags state of SDIO
SDIO_INT_FLAG_CCR CERR	command response received (CRC check failed) flag
SDIO_INT_FLAG_DTC RCERR	data block sent/received (CRC check failed) flag
SDIO_INT_FLAG_CMD TMOUT	command response timeout flag
SDIO_INT_FLAG_DTT MOUT	data timeout flag
SDIO_INT_FLAG_TXU RE	transmit FIFO underrun error occurs flag
SDIO_INT_FLAG_RXO RE	received FIFO overrun error occurs flag
SDIO_INT_FLAG_CMD RECV	command response received (CRC check passed) flag
SDIO_INT_FLAG_CMD SEND	command sent (no response required) flag
SDIO_INT_FLAG_DTE ND	data end (data counter, SDIO_DATACNT, is zero) flag
SDIO_INT_FLAG_STBI	start bit error in the bus flag

<i>TE</i>	
<i>SDIO_INT_FLAG_DTB LKEND</i>	data block sent/received (CRC check passed) flag
<i>SDIO_INT_FLAG_SDI OINT</i>	SD I/O interrupt received flag
<i>SDIO_INT_FLAG_ATA END</i>	CE-ATA command completion signal received (only for CMD61) flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the interrupt pending flags of SDIO */
sdio_interrupt_flag_clear(SDIO_INT_FLAG_DTEND);
```

### sdio\_readwait\_enable

The description of sdio\_readwait\_enable is shown as below:

**Table 3-703. Function sdio\_readwait\_enable**

<b>Function name</b>	sdio_readwait_enable
<b>Function prototype</b>	void sdio_readwait_enable(void);
<b>Function descriptions</b>	enable the read wait mode(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the read wait mode(SD I/O only) */
```

```
sdio_readwait_enable();
```

### sdio\_readwait\_disable

The description of sdio\_readwait\_disable is shown as below:

**Table 3-704. Function sdio\_readwait\_disable**

<b>Function name</b>	sdio_readwait_disable
<b>Function prototype</b>	void sdio_readwait_disable(void);
<b>Function descriptions</b>	disable the read wait mode(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the read wait mode(SD I/O only) */
```

```
sdio_readwait_disable();
```

### sdio\_stop\_readwait\_enable

The description of sdio\_stop\_readwait\_enable is shown as below:

**Table 3-705. Function sdio\_stop\_readwait\_enable**

<b>Function name</b>	sdio_stop_readwait_enable
<b>Function prototype</b>	void sdio_stop_readwait_enable(void);
<b>Function descriptions</b>	enable the function that stop the read wait process(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the function that stop the read wait process(SD I/O only) */
```

```
sdio_stop_readwait_enable();
```

### sdio\_stop\_readwait\_disable

The description of sdio\_stop\_readwait\_disable is shown as below:

**Table 3-706. Function sdio\_stop\_readwait\_disable**

Function name	sdio_stop_readwait_disable
Function prototype	void sdio_stop_readwait_disable(void);
Function descriptions	disable the function that stop the read wait process(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the function that stop the read wait process(SD I/O only) */
```

```
sdio_stop_readwait_disable();
```

### sdio\_readwait\_type\_set

The description of sdio\_readwait\_type\_set is shown as below:

**Table 3-707. Function sdio\_readwait\_type\_set**

Function name	sdio_readwait_type_set
---------------	------------------------

<b>Function prototype</b>	void sdio_readwait_type_set(uint32_t readwait_type);
<b>Function descriptions</b>	set the read wait type(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>readwait_type</b>	SD I/O read wait type
<i>SDIO_READWAITTYP E_CLK</i>	read wait control by stopping SDIO_CLK
<i>SDIO_READWAITTYP E_DAT2</i>	read wait control using SDIO_DAT[2]
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the read wait type(SD I/O only) */
sdio_readwait_type_set(uint32_t readwait_type);
```

### sdio\_operation\_enable

The description of sdio\_operation\_enable is shown as below:

**Table 3-708. Function sdio\_operation\_enable**

<b>Function name</b>	sdio_operation_enable
<b>Function prototype</b>	void sdio_operation_enable(void);
<b>Function descriptions</b>	enable the SD I/O mode specific operation(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-



Return value	
-	-

Example:

```
/* enable the SD I/O mode specific operation(SD I/O only) */
```

```
sdio_operation_enable();
```

### sdio\_operation\_disable

The description of sdio\_operation\_disable is shown as below:

**Table 3-709. Function sdio\_operation\_disable**

Function name	sdio_operation_disable
Function prototype	void sdio_operation_disable(void);
Function descriptions	disable the SD I/O mode specific operation(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the SD I/O mode specific operation(SD I/O only) */
```

```
void sdio_operation_disable();
```

### sdio\_suspend\_enable

The description of sdio\_suspend\_enable is shown as below:

**Table 3-710. Function sdio\_suspend\_enable**

Function name	sdio_suspend_enable
Function prototype	void sdio_suspend_enable(void);
Function descriptions	enable the SD I/O suspend operation(SD I/O only)

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

Example:

```
/* enable the SD I/O suspend operation(SD I/O only) */
```

```
sdio_suspend_enable();
```

### **sdio\_suspend\_disable**

The description of sdio\_suspend\_disable is shown as below:

**Table 3-711. Function sdio\_suspend\_disable**

<b>Function name</b>	sdio_suspend_disable
<b>Function prototype</b>	void sdio_suspend_disable(void);
<b>Function descriptions</b>	disable the SD I/O suspend operation(SD I/O only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the SD I/O suspend operation(SD I/O only) */
```

```
sdio_suspend_disable();
```

## sdio\_ceata\_command\_enable

The description of sdio\_ceata\_command\_enable is shown as below:

**Table 3-712. Function sdio\_ceata\_command\_enable**

<b>Function name</b>	sdio_ceata_command_enable
<b>Function prototype</b>	void sdio_ceata_command_enable(void);
<b>Function descriptions</b>	enable the CE-ATA command(CE-ATA only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the CE-ATA command(CE-ATA only) */
```

```
sdio_ceata_command_enable();
```

## sdio\_ceata\_command\_disable

The description of sdio\_ceata\_command\_disable is shown as below:

**Table 3-713. Function sdio\_ceata\_command\_disable**

<b>Function name</b>	sdio_ceata_command_disable
<b>Function prototype</b>	void sdio_ceata_command_disable(void);
<b>Function descriptions</b>	disable the CE-ATA command(CE-ATA only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* disable the CE-ATA command(CE-ATA only) */
```

```
sdio_ceata_command_disable();
```

### sdio\_ceata\_interrupt\_enable

The description of sdio\_ceata\_interrupt\_enable is shown as below:

**Table 3-714. Function sdio\_ceata\_interrupt\_enable**

Function name	sdio_ceata_interrupt_enable
Function prototype	void sdio_ceata_interrupt_enable(void);
Function descriptions	enable the CE-ATA interrupt(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CE-ATA interrupt(CE-ATA only) */
```

```
sdio_ceata_interrupt_enable();
```

### sdio\_ceata\_interrupt\_disable

The description of sdio\_ceata\_interrupt\_disable is shown as below:

**Table 3-715. Function sdio\_ceata\_interrupt\_disable**

Function name	sdio_ceata_interrupt_disable
Function prototype	void sdio_ceata_interrupt_disable(void);
Function descriptions	disable the CE-ATA interrupt(CE-ATA only)

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

Example:

```
/* disable the CE-ATA interrupt(CE-ATA only) */
```

```
sdio_ceata_interrupt_disable();
```

### **sdio\_ceata\_command\_completion\_enable**

The description of sdio\_ceata\_command\_completion\_enable is shown as below:

**Table 3-716. Function sdio\_ceata\_command\_completion\_enable**

<b>Function name</b>	sdio_ceata_command_completion_enable
<b>Function prototype</b>	void sdio_ceata_command_completion_enable(void);
<b>Function descriptions</b>	enable the CE-ATA command completion signal(CE-ATA only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the CE-ATA command completion signal(CE-ATA only) */
```

```
sdio_ceata_command_completion_enable();
```

## sdio\_ceata\_command\_completion\_disable

The description of sdio\_ceata\_command\_completion\_disable is shown as below:

**Table 3-717. Function sdio\_ceata\_command\_completion\_disable**

<b>Function name</b>	sdio_ceata_command_completion_disable
<b>Function prototype</b>	void sdio_ceata_command_completion_disable(void);
<b>Function descriptions</b>	disable the CE-ATA command completion signal(CE-ATA only)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the CE-ATA command completion signal(CE-ATA only) */
sdio_ceata_command_completion_disable();
```

## 3.24. 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 3-718. 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

Registers	Descriptions
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
I2S_ADD_CTL0	I2S_ADD control register 0
I2S_ADD_CTL1	I2S_ADD control register 1
I2S_ADD_STAT	I2S_ADD status register
I2S_ADD_DATA	I2S_ADD data register
I2S_ADD_CRCPOLY	I2S_ADD CRC polynomial register
I2S_ADD_RCRC	I2S_ADD receive CRC register
I2S_ADD_TCRC	I2S_ADD transmit CRC register
I2S_ADD_I2SCTL	I2S_ADD I2S control register
I2S_ADD_I2SPSC	I2S_ADD I2S clock prescaler register

### 3.24.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

**Table 3-719. SPI/I2S firmware function**

Function name	Function description
spi_i2s_deinit	reset SPI and I2S peripheral
spi_struct_para_init	initialize the parameters of SPI struct with the default values
spi_init	initialize SPI peripheral parameter
spi_enable	enable SPI
spi_disable	disable SPI
i2s_init	initialize I2S peripheral parameter
i2s_psc_config	configure I2S peripheral prescaler
i2s_enable	enable I2S
i2s_disable	disable I2S
spi_nss_output_enable	enable SPI NSS output function
spi_nss_output_disable	disable SPI NSS output function
spi_nss_internal_high	SPI NSS pin high level in software mode
spi_nss_internal_low	SPI NSS pin low level in software mode
spi_dma_enable	enable SPI DMA function
spi_dma_disable	disable SPI DMA function
spi_i2s_data_frame_format_config	configure SPI/I2S data frame format
spi_i2s_data_transmit	SPI transmit data
spi_i2s_data_receive	SPI receive data
spi_bidirectional_transfer_config	configure SPI bidirectional transfer direction
i2s_full_duplex_mode_config	configure i2s full duplex mode
spi_i2s_format_error_clear	clear TI mode format error flag status

Function name	Function description
spi_crc_polynomial_set	set SPI CRC polynomial
spi_crc_polynomial_get	get SPI CRC polynomial
spi_crc_on	turn on SPI CRC function
spi_crc_off	turn off SPI CRC function
spi_crc_next	SPI next data is CRC value
spi_crc_get	get SPI CRC send value or receive value
spi_crc_error_clear	clear SPI CRC error flag status
spi_ti_mode_enable	enable SPI TI mode
spi_ti_mode_disable	disable SPI TI mode
spi_quad_enable	enable quad wire SPI
spi_quad_disable	disable quad wire SPI
spi_quad_write_enable	enable quad wire SPI write
spi_quad_read_enable	enable quad wire SPI read
spi_quad_io23_output_enable	enable quad wire SPI_IO2 and SPI_IO3 pin output
spi_quad_io23_output_disable	disable quad wire SPI_IO2 and SPI_IO3 pin output
spi_i2s_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 3-720. spi\_parameter\_struct

Member name	Function description
device_mode	SPI master or slave (SPI_MASTER, SPI_SLAVE)
trans_mode	SPI transtype (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	SPI frame size (SPI_FRAME_SIZE_16BIT, SPI_FRAME_SIZE_8BIT)
nss	SPI NSS control by hardware or software (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	SPI big endian or little endian (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	SPI clock phase and polarity (SPI_CK_PL_LOW_PH_1EDGE, SPI_CK_PL_HIGH_PH_1EDGE, SPI_CK_PL_LOW_PH_2EDGE, SPI_CK_PL_HIGH_PH_2EDGE)
prescale	SPI prescale factor (SPI_PSC_n (n=2,4,8,16,32,64,128,256))



## spi\_i2s\_deinit

The description of spi\_i2s\_deinit is shown as below:

**Table 3-721. Function spi\_i2s\_deinit**

<b>Function name</b>	spi_i2s_deinit
<b>Function prototype</b>	void spi_i2s_deinit(uint32_t spi_periph);
<b>Function descriptions</b>	reset SPI and I2S peripheral
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI/I2S peripheral
<b>SPIx</b>	x=0,1,2,3,4,5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset SPI0 */
spi_i2s_deinit(SPI0);
```

## spi\_struct\_para\_init

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

**Table 3-722. Function spi\_struct\_para\_init**

<b>Function name</b>	spi_struct_para_init
<b>Function prototype</b>	void spi_struct_para_init(spi_parameter_struct* spi_struct);
<b>Function descriptions</b>	initialize the parameters of SPI struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>*spi_struct</b>	a spi_parameter_struct address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize the parameters of SPI */
spi_parameter_struct spi_init_struct;
spi_struct_para_init(&spi_init_struct);
```

## spi\_init

The description of spi\_init is shown as below:

**Table 3-723. Function spi\_init**

<b>Function name</b>	spi_init
<b>Function prototype</b>	void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
<b>Function descriptions</b>	initialize SPI peripheral parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1,2,3,4,5
<b>Input parameter{in}</b>	
<b>spi_struct</b>	SPI parameter initialization struct, the structure members can refer to members of the structure <a href="#">Table 3-720. 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_CK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss             = SPI_NSS_SOFT;
spi_init_struct.prescale        = SPI_PSC_8;
spi_init_struct.endian          = SPI_ENDIAN_MSB;

spi_init(SPI0, &spi_init_struct);

```

## spi\_enable

The description of spi\_enable is shown as below:

**Table 3-724. 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=0,1,2,3,4,5
<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 3-725. 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=0,1,2,3,4,5
<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 3-726. Function i2s\_init**

<b>Function name</b>	i2s_init
<b>Function prototype</b>	void i2s_init(uint32_t spi_periph, uint32_t i2s_mode, uint32_t i2s_standard, uint32_t i2s_ckpl);

<b>Function descriptions</b>	initialize I2S peripheral parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<i>SPIx</i>	x=1,2
<b>Input parameter{in}</b>	
<b>i2s_mode</b>	I2S operation mode
<i>I2S_MODE_SLAVETX</i>	I2S slave transmit mode
<i>I2S_MODE_SLAVERX</i>	I2S slave receive mode
<i>I2S_MODE_MASTERTX</i> <i>X</i>	I2S master transmit mode
<i>I2S_MODE_MASTERRX</i> <i>X</i>	I2S master receive mode
<b>Input parameter{in}</b>	
<b>i2s_standard</b>	I2S standard
<i>I2S_STD_PHILLIPS</i>	I2S phillips standard
<i>I2S_STD_MSB</i>	I2S MSB standard
<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCMSHORT</i>	I2S PCM short standard
<i>I2S_STD_PCMLONG</i>	I2S PCM long standard
<b>Input parameter{in}</b>	
<b>i2s_ckpl</b>	I2S idle state clock polarity
<i>I2S_CKPL_LOW</i>	I2S clock polarity low level
<i>I2S_CKPL_HIGH</i>	I2S clock polarity high level
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize I2S1 */
```

```
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

### i2s\_psc\_config

The description of i2s\_psc\_config is shown as below:

**Table 3-727. 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_i2s_clock_config/ rcu_osci_on/ rcu_osci_stab_wait
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<b>SPIx</b>	x=1,2
<b>Input parameter{in}</b>	
<b>i2s_audiosample</b>	I2S audio sample rate
<b>I2S_AUDIOSAMPLE_8K</b>	audio sample rate is 8KHz
<b>I2S_AUDIOSAMPLE_11K</b>	audio sample rate is 11KHz
<b>I2S_AUDIOSAMPLE_16K</b>	audio sample rate is 16KHz
<b>I2S_AUDIOSAMPLE_22K</b>	audio sample rate is 22KHz
<b>I2S_AUDIOSAMPLE_32K</b>	audio sample rate is 32KHz
<b>I2S_AUDIOSAMPLE_44K</b>	audio sample rate is 44KHz
<b>I2S_AUDIOSAMPLE_48K</b>	audio sample rate is 48KHz
<b>I2S_AUDIOSAMPLE_96K</b>	audio sample rate is 96KHz
<b>I2S_AUDIOSAMPLE_192K</b>	audio sample rate is 192KHz
<b>Input parameter{in}</b>	
<b>i2s_frameformat</b>	I2S data length and channel length
<b>I2S_FRAMEFORMAT_DT16B_CH16B</b>	I2S data length is 16 bit and channel length is 16 bit
<b>I2S_FRAMEFORMAT_DT16B_CH32B</b>	I2S data length is 16 bit and channel length is 32 bit
<b>I2S_FRAMEFORMAT_DT24B_CH32B</b>	I2S data length is 24 bit and channel length is 32 bit
<b>I2S_FRAMEFORMAT_DT32B_CH32B</b>	I2S data length is 32 bit and channel length is 32 bit
<b>Input parameter{in}</b>	
<b>i2s_mckout</b>	I2S master clock output
<b>I2S_MCKOUT_ENABLER</b>	I2S master clock output enable
<b>I2S_MCKOUT_DISABLE</b>	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 3-728. Function i2s\_enable**

<b>Function name</b>	i2s_enable
<b>Function prototype</b>	void i2s_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable I2S
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<i>SPIx</i>	x=1,2
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2S1*/
```

```
i2s_enable(SPI1);
```

## i2s\_disable

The description of i2s\_disable is shown as below:

**Table 3-729. Function i2s\_disable**

<b>Function name</b>	i2s_disable
<b>Function prototype</b>	void i2s_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable I2S
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S peripheral
<i>SPIx</i>	x=1,2
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* disable I2S1*/
```

```
i2s_disable(SPI1);
```

### **spi\_nss\_output\_enable**

The description of spi\_nss\_output\_enable is shown as below:

**Table 3-730. Function spi\_nss\_output\_enable**

Function name	spi_nss_output_enable
Function prototype	void spi_nss_output_enable(uint32_t spi_periph);
Function descriptions	enable SPI NSS output function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
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 3-731. Function spi\_nss\_output\_disable**

Function name	spi_nss_output_disable
Function prototype	void spi_nss_output_disable(uint32_t spi_periph);
Function descriptions	disable SPI NSS output function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
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 3-732. 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=0,1,2,3,4,5
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 3-733. 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=0,1,2,3,4,5
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 3-734. Function spi\_dma\_enable**

Function name	spi_dma_enable
Function prototype	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
Function descriptions	enable SPI DMA function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Input parameter{in}	
dma	SPI DMA mode
SPI_DMA_TRANSMIT	SPI transmit data use DMA
SPI_DMA_RECEIVE	SPI receive data use DMA
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 transmit data DMA function */
```

```
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

### spi\_dma\_disable

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

**Table 3-735. Function spi\_dma\_disable**

Function name	spi_dma_disable
Function prototype	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
Function descriptions	disable SPI DMA function
Precondition	-
The called functions	-
Input parameter{in}	

<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
<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 3-736. Function spi\_i2s\_data\_frame\_format\_config**

<b>Function name</b>	spi_i2s_data_frame_format_config
<b>Function prototype</b>	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
<b>Function descriptions</b>	configure SPI/I2S data frame format
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
<b>Input parameter{in}</b>	
<b>frame_format</b>	SPI frame size
<i>SPI_FRAME_SIZE_16BIT</i>	SPI frame size is 16 bits
<i>SPI_FRAME_SIZE_8BIT</i>	SPI frame size is 8 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SPI1/I2S1 data frame format size is 16 bits */
```

```
spi_i2s_data_frame_format_config(SPI1, SPI_FRAME_SIZE_16BIT);
```

## spi\_i2s\_data\_transmit

The description of spi\_i2s\_data\_transmit is shown as below:

**Table 3-737. Function spi\_i2s\_data\_transmit**

<b>Function name</b>	spi_i2s_data_transmit
<b>Function prototype</b>	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
<b>Function descriptions</b>	SPI transmit data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
<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 spi_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 3-738. 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=0,1,2,3,4,5
<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 3-739. 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=0,1,2,3,4,5
<b>Input parameter{in}</b>	
<b>transfer_direction</b>	SPI transfer direction
<i>SPI_BIDIRECTIONAL_TRANSMIT</i>	SPI work in transmit-only mode
<i>SPI_BIDIRECTIONAL_RECEIVE</i>	SPI work in receive-only mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* SPI0 works in transmit-only mode */

spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);

```

### i2s\_full\_duplex\_mode\_config

The description of i2s\_full\_duplex\_mode\_config is shown as below:

**Table 3-740. Function i2s\_full\_duplex\_mode\_config**

<b>Function name</b>	i2s_full_duplex_mode_config
<b>Function prototype</b>	void i2s_full_duplex_mode_config (uint32_t i2s_add_periph, uint32_t i2s_mode, uint32_t i2s_standard, uint32_t i2s_ckpl, uint32_t frameformat);
<b>Function descriptions</b>	configure i2s full duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>i2s_add_periph</b>	I2Sx_ADDperipheral
<i>I2Sx_ADD</i>	x=1,2
Input parameter{in}	
<b>i2s_mode</b>	i2s mode
<i>I2S_MODE_SLAVETX</i>	I2S slave transmit mode
<i>I2S_MODE_SLAVERX</i>	I2S slave receive mode
<i>I2S_MODE_MASTERT</i> <i>X</i>	I2S master transmit mode
<i>I2S_MODE_MASTERR</i> <i>X</i>	I2S master receive mode
Input parameter{in}	
<b>i2s_standard</b>	i2s standard
<i>I2S_STD_PHILLIPS</i>	I2S phillips standard
<i>I2S_STD_MSB</i>	I2S MSB standard
<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCMSHORT</i>	I2S PCM short standard
<i>I2S_STD_PCMLONG</i>	I2S PCM long standard
Input parameter{in}	
<b>i2s_ckpl</b>	i2s idle state clock polarity
<i>I2S_CKPL_LOW</i>	The idle state of I2S_CK is low level
<i>I2S_CKPL_HIGH</i>	The idle state of I2S_CK is high level
Input parameter{in}	
<b>i2s_frameformat</b>	i2s data length and channel length
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S data length is 16 bit and channel length is 16 bit
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S data length is 16 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S data length is 24 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S data length is 32 bit and channel length is 32 bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure i2s full duplex mode */
```

```
i2s_full_duplex_mode_config(I2S1_ADD,I2S_MODE_SLAVETX,I2S_STD_LSB,I2S_CKPL_
LOW,I2S_FRAMEFORMAT_DT16B_CH16B);
```

## spi\_i2s\_format\_error\_clear

The description of spi\_i2s\_format\_error\_clear is shown as below:

**Table 3-741. Function spi\_i2s\_format\_error\_clear**

<b>Function name</b>	spi_i2s_format_error_clear
<b>Function prototype</b>	void spi_i2s_format_error_clear(uint32_t spi_periph, uint32_t flag);
<b>Function descriptions</b>	clear TI mode format error flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
<b>Input parameter{in}</b>	
<b>flag</b>	SPI/I2S frame format error flag
<i>SPI_FLAG_FERR</i>	SPI TI mode frame format error
<i>I2S_FLAG_FERR</i>	I2S frame format error
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear SPI0 TI mode format error flag */
spi_i2s_format_error_clear(SPI0, SPI_FLAG_FERR);
```

## spi\_crc\_polynomial\_set

The description of spi\_crc\_polynomial\_set is shown as below:

**Table 3-742. 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=0,1,2,3,4,5
<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 */
uint16_t CRC_VALUE = 0x5050;

spi_crc_polynomial_set(SPI0,CRC_VALUE);
```

### spi\_crc\_polynomial\_get

The description of spi\_crc\_polynomial\_get is shown as below:

**Table 3-743. Function spi\_crc\_polynomial\_get**

<b>Function name</b>	spi_crc_polynomial_get
<b>Function prototype</b>	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
<b>Function descriptions</b>	get SPI CRC polynomial
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	16-bit CRC polynomial (0-0xFFFF)

Example:

```
/* get SPI0 CRC polynomial */
uint16_t crc_val;

crc_val = spi_crc_polynomial_get(SPI0);
```

### spi\_crc\_on

The description of spi\_crc\_on is shown as below:

**Table 3-744. Function spi\_crc\_on**

<b>Function name</b>	spi_crc_on
<b>Function prototype</b>	void spi_crc_on(uint32_t spi_periph);
<b>Function descriptions</b>	turn on SPI CRC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5

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 3-745. Function spi\_crc\_off**

Function name	spi_crc_off
Function prototype	void spi_crc_off(uint32_t spi_periph);
Function descriptions	turn off SPI CRC function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn off SPI0 CRC function */
```

```
spi_crc_off(SPI0);
```

## spi\_crc\_next

The description of spi\_crc\_next is shown as below:

**Table 3-746. Function spi\_crc\_next**

Function name	spi_crc_next
Function prototype	void spi_crc_next(uint32_t spi_periph);
Function descriptions	SPI next data is CRC value
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5



Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 next data is CRC value */
```

```
spi_crc_next(SPI0);
```

## spi\_crc\_get

The description of spi\_crc\_get is shown as below:

**Table 3-747. Function spi\_crc\_get**

Function name	spi_crc_get
Function prototype	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t spi_crc);
Function descriptions	get SPI CRC send value or receive value
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Input parameter{in}	
spi_crc	SPI crc value
SPI_CRC_TX	get transmit crc value
SPI_CRC_RX	get receive crc value
Output parameter{out}	
-	-
Return value	
uint16_t	16-bit CRC value (0-0xFFFF)

Example:

```
/* get SPI0 CRC send value */
```

```
uint16_t crc_val;
```

```
crc_val = spi_crc_get(SPI0, SPI_CRC_TX);
```

## spi\_crc\_error\_clear

The description of spi\_crc\_error\_clear is shown as below:

**Table 3-748. Function spi\_crc\_error\_clear**

Function name	spi_crc_error_clear
Function prototype	void spi_crc_error_clear(uint32_t spi_periph);
Function descriptions	clear SPI CRC error flag status

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
<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 3-749. 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=0,1,2,3,4,5
<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 3-750. 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=0,1,2,3,4,5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

### spi\_quad\_enable

The description of spi\_quad\_enable is shown as below:

**Table 3-751. 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=5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI5 quad wire mode */
spi_quad_enable(SPI5);
```

### spi\_quad\_disable

The description of spi\_quad\_disable is shown as below:

**Table 3-752. 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=5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI5 quad wire mode */
spi_quad_disable(SPI5);
```

### spi\_quad\_write\_enable

The description of spi\_quad\_write\_enable is shown as below:

**Table 3-753. 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=5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI5 quad wire write */
spi_quad_write_enable(SPI5);
```

### spi\_quad\_read\_enable

The description of spi\_quad\_read\_enable is shown as below:

**Table 3-754. 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=5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI5 quad wire read */
spi_quad_read_enable(SPI5);
```

### spi\_quad\_io23\_output\_enable

The description of spi\_quad\_io23\_output\_enable is shown as below:

**Table 3-755. Function spi\_quad\_io23\_output\_enable**

<b>Function name</b>	spi_quad_io23_output_enable
<b>Function prototype</b>	void spi_quad_io23_output_enable(uint32_t spi_periph);
<b>Function descriptions</b>	enable SPI_IO2 and SPI_IO3 pin output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI5 SPI_IO2 and SPI_IO3 pin output */
spi_quad_io23_output_enable(SPI5);
```

### spi\_quad\_io23\_output\_disable

The description of spi\_quad\_io23\_output\_disable is shown as below:

**Table 3-756. Function spi\_quad\_io23\_output\_disable**

<b>Function name</b>	spi_quad_io23_output_disable
<b>Function prototype</b>	void spi_quad_io23_output_disable(uint32_t spi_periph);
<b>Function descriptions</b>	disable SPI_IO2 and SPI_IO3 pin output

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=5
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI5 SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_disable(SPI5);
```

### spi\_i2s\_flag\_get

The description of spi\_i2s\_flag\_get is shown as below:

**Table 3-757. 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 spi_i2s_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=0,1,2,3,4,5
<b>Input parameter{in}</b>	
<b>spi_i2s_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_FLAG_RXORERR</i>	receive overrun error flag
<i>SPI_FLAG_CONFERR</i>	mode config error flag
<i>SPI_FLAG_CRCERR</i>	CRC error flag
<i>SPI_FLAG_FERR</i>	format error flag
<i>I2S_FLAG_TBE</i>	transmit buffer empty flag
<i>I2S_FLAG_RBNE</i>	receive buffer not empty flag
<i>I2S_FLAG_TRANS</i>	transmit on-going flag
<i>I2S_FLAG_RXORERR</i>	overrun error flag
<i>I2S_FLAG_TXURERR</i>	underrun error flag
<i>I2S_FLAG_CH</i>	channel side flag
<i>I2S_FLAG_FERR</i>	format error flag

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

Example:

```
/* get SPI0 transmit buffer empty flag status */
while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

### spi\_i2s\_interrupt\_enable

The description of spi\_i2s\_interrupt\_enable is shown as below:

**Table 3-758. Function spi\_i2s\_interrupt\_enable**

Function name	spi_i2s_interrupt_enable
Function prototype	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t spi_i2s_int);
Function descriptions	enable SPI and I2S interrupt
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Input parameter{in}	
spi_i2s_int	SPI/I2S interrupt
SPI_I2S_INT_TBE	transmit buffer empty interrupt
SPI_I2S_INT_RBNE	receive buffer not empty interrupt
SPI_I2S_INT_ERR	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 transmit buffer empty interrupt */
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

### spi\_i2s\_interrupt\_disable

The description of spi\_i2s\_interrupt\_disable is shown as below:

**Table 3-759. Function spi\_i2s\_interrupt\_disable**

Function name	spi_i2s_interrupt_disable
---------------	---------------------------

<b>Function prototype</b>	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t spi_i2s_int);
<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=0,1,2,3,4,5
<b>Input parameter{in}</b>	
<b>spi_i2s_int</b>	SPI/I2S interrupt
<i>SPI_I2S_INT_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

### spi\_i2s\_interrupt\_flag\_get

The description of spi\_i2s\_interrupt\_flag\_get is shown as below:

**Table 3-760. Function spi\_i2s\_interrupt\_flag\_get**

<b>Function name</b>	spi_i2s_interrupt_flag_get
<b>Function prototype</b>	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t spi_i2s_int);
<b>Function descriptions</b>	get SPI and I2S interrupt status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
<b>Input parameter{in}</b>	
<b>spi_i2s_int</b>	SPI/I2S interrupt flag status
<i>SPI_I2S_INT_FLAG_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_FLAG_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_FLAG_RXORERR</i>	overrun interrupt



<i>SPI_INT_FLAG_CONFERR</i>	config error interrupt
<i>SPI_INT_FLAG_CRCERR</i>	CRC error interrupt
<i>I2S_INT_FLAG_TXURERR</i>	underrun error interrupt
<i>I2S_INT_FLAG_FERR</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. SYSCFG

The SYSCFG 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

SYSCFG registers are listed in the table shown as below:

**Table 3-761. SYSCFG registers**

Registers	Descriptions
SYSCFG_CFG0	Configuration register 0
SYSCFG_CFG1	Configuration register 1
SYSCFG_EXTISS0	EXTI sources selection register 0
SYSCFG_EXTISS1	EXTI sources selection register 1
SYSCFG_EXTISS2	EXTI sources selection register 2
SYSCFG_EXTISS3	EXTI sources selection register 3
SYSCFG_CPSCCTL	I/O compensation control register

### 3.25.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-762.SYSCFG firmware function**

Function name	Function description
syscfg_deinit	deinit syscfg module
syscfg_bootmode_config	configure the boot mode
syscfg_fmc_swap_config	configure FMC memory mapping swap
syscfg_exmc_swap_config	configure the EXMC swap
syscfg_exti_line_config	configure the GPIO pin as EXTI Line
syscfg_enet_phy_interface_config	configure the PHY interface for the ethernet MAC
syscfg_compensation_config	configure the I/O compensation cell
syscfg_flag_get	check the I/O compensation cell is ready or not

#### syscfg\_deinit

The description of syscfg\_deinit is shown as below:

**Table 3-763. Function syscfg\_deinit**

Function name	syscfg_deinit
Function prototype	void syscfg_deinit(void);
Function descriptions	deinit syscfg module
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset SYSCFG */
```

```
syscfg_deinit();
```

## syscfg\_bootmode\_config

The description of syscfg\_bootmode\_config is shown as below:

**Table 3-764. Function syscfg\_bootmode\_config**

<b>Function name</b>	syscfg_bootmode_config
<b>Function prototype</b>	void syscfg_bootmode_config(uint8_t syscfg_bootmode);
<b>Function descriptions</b>	configure the boot mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_bootmode</b>	selects the memory remapping
SYSCFG_BOOTMODE_FLASH	main flash memory (0x08000000~0x083BFFFF) is mapped at address 0x00000000
SYSCFG_BOOTMODE_BOOTLOADER	boot loader (0x1FFF0000 - 0x1FFF77FF) is mapped at address 0x00000000
SYSCFG_BOOTMODE_EXMC_SRAM	SRAM/NOR 0 and 1 of EXMC (0x60000000~0x67FFFFFF) is mapped at address 0x00000000
SYSCFG_BOOTMODE_SRAM	SRAM0 of on-chip SRAM (0x20000000~0x2001BFFF) is mapped at address 0x00000000
SYSCFG_BOOTMODE_EXMC_SDRAM	SDRAM bank0 of EXMC (0xC0000000~0xC7FFFFFF) is mapped at address 0x00000000
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure boot from main flash */
syscfg_bootmode_config(SYSCFG_BOOTMODE_FLASH);
```

## syscfg\_fmc\_swap\_config

The description of syscfg\_fmc\_swap\_config is shown as below:

Table 3-765. Function syscfg\_fmc\_swap\_config

Function name	syscfg_fmc_swap_config
Function prototype	void syscfg_fmc_swap_config(uint32_t syscfg_fmc_swap);
Function descriptions	FMC memory mapping swap
Precondition	-
The called functions	-
Input parameter{in}	
syscfg_fmc_swap	selects the interal flash bank swapping
SYSCFG_FMC_SWP_BANK0	bank 0 is mapped at address 0x08000000 and bank 1 is mapped at address 0x08100000
SYSCFG_FMC_SWP_BANK1	bank 1 is mapped at address 0x08000000 and bank 0 is mapped at address 0x08100000
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* FMC memory bank 0 is mapped at address 0x08000000 and bank 1 is mapped at address 0x08100000 */
```

```
syscfg_fmc_swap_config(SYSCFG_FMC_SWP_BANK0);
```

### syscfg\_exmc\_swap\_config

The description of syscfg\_exmc\_swap\_config is shown as below:

Table 3-766. Function syscfg\_exmc\_swap\_config

Function name	syscfg_exmc_swap_config
Function prototype	void syscfg_exmc_swap_config(uint32_t syscfg_exmc_swap);
Function descriptions	EXMC memory mapping swap
Precondition	-
The called functions	-
Input parameter{in}	
syscfg_exmc_swap	selects the memories in EXMC swapping

<code>SYSCFG_EXMC_SWP_ENABLE</code>	SDRAM bank 0 and bank 1 are swapped with NAND bank 1 and PC card
<code>SYSCFG_EXMC_SWP_DISABLE</code>	no memory mapping swap
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable EXMC memory mapping swap */
```

```
syscfg_exmc_swap_config(SYSCFG_EXMC_SWP_ENABLE);
```

### syscfg\_exti\_line\_config

The description of syscfg\_exti\_line\_config is shown as below:

**Table 3-767. Function syscfg\_exti\_line\_config**

<b>Function name</b>	syscfg_exti_line_config
<b>Function prototype</b>	void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);
<b>Function descriptions</b>	configure the GPIO pin as EXTI Line
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>exti_port</b>	specify the GPIO port used in EXTI
<code>EXTI_SOURCE_GPIOx</code>	x = A,B,C,D,E,F,G,H,I
<b>Input parameter{in}</b>	
<b>exti_pin</b>	specify the EXTI line
<code>EXTI_SOURCE_PINx</code>	x = 0..15
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PA0 pin as EXTI Line */
syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN0);
```

### syscfg\_enet\_phy\_interface\_config

The description of syscfg\_enet\_phy\_interface\_config is shown as below:

**Table 3-768. Function syscfg\_enet\_phy\_interface\_config**

<b>Function name</b>	syscfg_enet_phy_interface_config
<b>Function prototype</b>	void syscfg_enet_phy_interface_config(uint32_t syscfg_enet_phy_interface);
<b>Function descriptions</b>	configure the PHY interface for the ethernet MAC
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_enet_phy_interface</b>	specifies the media interface mode
<i>SYSCFG_ENET_PHY_MII</i>	MII mode is selected
<i>SYSCFG_ENET_PHY_RMII</i>	RMII mode is selected
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the MII PHY interface for the ethernet MAC */
syscfg_enet_phy_interface_config(SYSCFG_ENET_PHY_MII);
```

### syscfg\_compensation\_config

The description of syscfg\_compensation\_config is shown as below:

**Table 3-769. Function syscfg\_compensation\_config**

<b>Function name</b>	syscfg_compensation_config
----------------------	----------------------------

<b>Function prototype</b>	void syscfg_compensation_config(uint32_t syscfg_compensation);
<b>Function descriptions</b>	configure the I/O compensation cell
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_compensation</b>	specifies the I/O compensation cell mode
SYSCFG_COMPENSATION_ENABLE	I/O compensation cell is enabled
SYSCFG_COMPENSATION_DISABLE	I/O compensation cell is disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the I/O compensation cell function */
```

```
syscfg_compensation_config(SYSCFG_COMPENSATION_ENABLE);
```

## syscfg\_flag\_get

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

**Table 3-770. Function syscfg\_flag\_get**

<b>Function name</b>	syscfg_flag_get
<b>Function prototype</b>	FlagStatus syscfg_flag_get(void);
<b>Function descriptions</b>	checks whether the I/O compensation cell ready flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-

Return value	
FlagStatus	SET or RESET

Example:

```
/* check whether the I/O compensation cell ready flag is set or not */
```

```
if(RESET != syscfg_flag_get());
```

## 3.26. TIMER

The timers have a 16-bit counter that can be used as an unsigned counter and supports both input capture and output compare. Timers (TIMERx) are divided into five sorts: advanced timer (TIMERx, x=0, 7), general level0 timer (TIMERx, x=1, 2, 3, 4), general level1 timer (TIMERx, x=8, 11), general level2 timer (TIMERx, x=9, 10, 12, 13), Basic timer (TIMERx, x=5, 6). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.26.1](#), the TIMER firmware functions are introduced in chapter [3.26.2](#).

### 3.26.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

**Table 3-771. TIMERx Registers**

Registers	Descriptions
TIMER_CTL0	Control register 0
TIMER_CTL1	Control register 1
TIMER_SMCFG	Slave mode configuration register
TIMER_DMAINTEN	DMA and interrupt enable register
TIMER_INTF	Interrupt flag register
TIMER_SWEVG	Software event generation register
TIMER_CHCTL0	Channel control register 0
TIMER_CHCTL1	Channel control register 1
TIMER_CHCTL2	Channel control register 2
TIMER_CNT	Counter register
TIMER_PSC	Prescaler register
TIMER_CAR	Counter auto reload register
TIMER_CREP	Counter repetition register



Registers	Descriptions
TIMER_CH0CV	Channel 0 capture/compare value register
TIMER_CH1CV	Channel 1 capture/compare value register
TIMER_CH2CV	Channel 2 capture/compare value register
TIMER_CH3CV	Channel 3 capture/compare value register
TIMER_CCHP	Channel complementary protection register
TIMER_DMACFG	DMA configuration register
TIMER_DMATB	DMA transfer buffer register
TIMER_IRMP	TIMER input remap register
TIMER_CFG	Configuration register

### 3.26.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-772.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

Function name	Function description
timer_repetition_value_config	configure TIMER repetition register value
timer_autoreload_value_config	configure TIMER autoreload register value
timer_counter_value_config	configure TIMER counter register value
timer_counter_read	read TIMER counter value
timer_prescaler_read	read TIMER prescaler value
timer_single_pulse_mode_config	configure TIMER single pulse mode
timer_update_source_config	configure TIMER update source
timer_dma_enable	enable the TIMER DMA
timer_dma_disable	disable the TIMER DMA
timer_channel_dma_request_source_select	channel DMA request source selection
timer_dma_transfer_config	configure the TIMER DMA transfer
timer_event_software_generate	software generate events
timer_break_struct_para_init	initialize the parameters of TIMER break parameter struct with the default values
timer_break_config	configure TIMER break function
timer_break_enable	enable TIMER break function
timer_break_disable	disable TIMER break function
timer_automatic_output_enable	enable TIMER output automatic function
timer_automatic_output_disable	disable TIMER output automatic function
timer_primary_output_config	configure TIMER primary output function
timer_channel_control_shadow_config	channel capture/compare control shadow register enable
timer_channel_control_shadow_update_config	configure TIMER channel control shadow register update control
timer_channel_output_struct_para_init	initialize the parameters of TIMER channel output parameter struct with the default values
timer_channel_output_config	configure TIMER channel output function
timer_channel_output_mode_config	configure TIMER channel output compare mode

Function name	Function description
timer_channel_output_pulse_value_config	configure TIMER channel output pulse value
timer_channel_output_shadow_config	configure TIMER channel output shadow function
timer_channel_output_fast_config	configure TIMER channel output fast function
timer_channel_output_clear_config	configure TIMER channel output clear function
timer_channel_output_polarity_config	configure TIMER channel output polarity
timer_channel_complementary_output_polarity_config	configure TIMER channel complementary output polarity
timer_channel_output_state_config	configure TIMER channel enable state
timer_channel_complementary_output_state_config	configure TIMER channel complementary output enable state
timer_channel_input_struct_para_init	initialize the parameters of TIMER channel input parameter struct with the default values
timer_input_capture_config	configure TIMER input capture parameter
timer_channel_input_capture_prescaler_config	configure TIMER channel input capture prescaler value
timer_channel_capture_value_register_read	read TIMER channel capture compare register value
timer_input_pwm_capture_config	configure TIMER input pwm capture function
timer_hall_mode_config	configure TIMER hall sensor mode
timer_input_trigger_source_select	select TIMER input trigger source
timer_master_output_trigger_source_select	select TIMER master mode output trigger source
timer_slave_mode_select	select TIMER slave mode
timer_master_slave_mode_config	configure TIMER master slave mode
timer_external_trigger_config	configure TIMER external trigger input
timer_quadrature_decoder_mode_config	configure TIMER quadrature decoder mode
timer_internal_clock_config	configure TIMER internal clock mode

Function name	Function description
timer_internal_trigger_as_external_clock_config	configure TIMER the internal trigger as external clock input
timer_external_trigger_as_external_clock_config	configure TIMER the external trigger as external clock input
timer_external_clock_mode0_config	configure TIMER the external clock mode 0
timer_external_clock_mode1_config	configure TIMER the external clock mode 1
timer_external_clock_mode1_disable	disable TIMER the external clock mode 1
timer_write_chxval_register_config	configure TIMER write CHxVAL register selection
timer_output_value_selection_config	configure TIMER output value selection
timer_channel_remap_config	configure TIMER channel remap function
timer_flag_get	get TIMER flags
timer_flag_clear	clear TIMER flags
timer_interrupt_enable	enable the TIMER interrupt
timer_interrupt_disable	disable the TIMER interrupt
timer_interrupt_flag_get	get timer interrupt flag
timer_interrupt_flag_clear	clear TIMER interrupt flag

### Structure timer\_parameter\_struct

**Table 3-773. Structure timer\_parameter\_struct**

Member name	Function description
prescaler	prescaler value (0~65535)
alignedmode	aligned mode (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)
counterdirection	counter direction (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)
period	period value (0~65535)
clockdivision	clock division value (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)
repetitioncounter	the counter repetition value (0~255)

## Structure timer\_break\_parameter\_struct

**Table 3-774. Structure timer\_break\_parameter\_struct**

Member name	Function description
runoffstate	run mode off-state (TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE)
ideloffstate	idle mode off-state (TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE)
deadtime	dead time (0~255)
breakpolarity	break polarity (TIMER_BREAK_POLARITY_LOW, TIMER_BREAK_POLARITY_HIGH)
outputautostate	output automatic enable (TIMER_OUTAUTO_ENABLE, TIMER_OUTAUTO_DISABLE)
protectmode	complementary register protect control (TIMER_CCHP_PROT_OFF, TIMER_CCHP_PROT_0, TIMER_CCHP_PROT_1, TIMER_CCHP_PROT_2)
breakstate	break enable (TIMER_BREAK_ENABLE, TIMER_BREAK_DISABLE)

## Structure timer\_oc\_parameter\_struct

**Table 3-775. Structure timer\_oc\_parameter\_struct**

Member name	Function description
outputstate	channel output state (TIMER_CCX_ENABLE, TIMER_CCX_DISABLE)
outputnstate	channel complementary output state (TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE)
ocpolarity	channel output polarity (TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW)
ocnpolarity	channel complementary output polarity (TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW)
ocidlestate	idle state of channel output (TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH)
ocnidlestate	idle state of channel complementary output (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH)

## Structure timer\_ic\_parameter\_struct

**Table 3-776. Structure timer\_ic\_parameter\_struct**

Member name	Function description
icpolarity	channel input polarity (TIMER_IC_POLARITY_RISING,

Member name	Function description
	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 3-777. Function timer\_deinit**

<b>Function name</b>	timer_deinit
<b>Function prototype</b>	void timer_deinit(uint32_t timer_periph);
<b>Function descriptions</b>	deinit a TIMER
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset TIMER0 */
timer_deinit (TIMER0);
```

## timer\_struct\_para\_init

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

**Table 3-778. Function timer\_struct\_para\_init**

<b>Function name</b>	timer_struct_para_init
----------------------	------------------------

<b>Function prototype</b>	void timer_struct_para_init(timer_parameter_struct* initpara);
<b>Function descriptions</b>	initialize the parameters of TIMER init parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>initpara</b>	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-773. 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 3-779. Function timer\_init**

<b>Function name</b>	timer_init
<b>Function prototype</b>	void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);
<b>Function descriptions</b>	initialize TIMER counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>initpara</b>	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-773. Structure timer parameter struct</a> .

Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize TIMER0 */

timer_parameter_struct timer_initpara;

timer_initpara.prescaler      = 107;

timer_initpara.alignedmode    = TIMER_COUNTER_EDGE;

timer_initpara.counterdirection = TIMER_COUNTER_UP;

timer_initpara.period         = 999;

timer_initpara.clockdivision   = TIMER_CKDIV_DIV1;

timer_initpara.repetitioncounter = 1;

timer_init(TIMER0,&timer_initpara);

```

## timer\_enable

The description of timer\_enable is shown as below:

**Table 3-780. Function timer\_enable**

Function name	timer_enable
Function prototype	void timer_enable(uint32_t timer_periph);
Function descriptions	enable a timer
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..13)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-



Example:

```
/* enable TIMER0 */
timer_enable (TIMER0);
```

### timer\_disable

The description of timer\_disable is shown as below:

**Table 3-781. Function timer\_disable**

<b>Function name</b>	timer_disable
<b>Function prototype</b>	void timer_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable a timer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 */
timer_disable (TIMER0);
```

### timer\_auto\_reload\_shadow\_enable

The description of timer\_auto\_reload\_shadow\_enable is shown as below:

**Table 3-782. Function timer\_auto\_reload\_shadow\_enable**

<b>Function name</b>	timer_auto_reload_shadow_enable
<b>Function prototype</b>	void timer_auto_reload_shadow_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable the auto reload shadow function
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable (TIMER0);
```

### timer\_auto\_reload\_shadow\_disable

The description of timer\_auto\_reload\_shadow\_disable is shown as below:

**Table 3-783. Function timer\_auto\_reload\_shadow\_disable**

<b>Function name</b>	timer_auto_reload_shadow_disable
<b>Function prototype</b>	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable the auto reload shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable (TIMER0);
```

## timer\_update\_event\_enable

The description of timer\_update\_event\_enable is shown as below:

**Table 3-784. Function timer\_update\_event\_enable**

<b>Function name</b>	timer_update_event_enable
<b>Function prototype</b>	void timer_update_event_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable the update event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 the update event */
timer_update_event_enable (TIMER0);
```

## timer\_update\_event\_disable

The description of timer\_update\_event\_disable is shown as below:

**Table 3-785. 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..13)</i>	TIMER peripheral selection

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 the update event */
```

```
timer_update_event_disable (TIMER0);
```

### timer\_counter\_alignment

The description of timer\_counter\_alignment is shown as below:

**Table 3-786. Function timer\_counter\_alignment**

Function name	timer_counter_alignment
Function prototype	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);
Function descriptions	set TIMER counter alignment mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMEx(x=0..4,7..13)	TIMER peripheral selection
Input parameter{in}	
aligned	alignment mode
TIMER_COUNTER_EDGE	No center-aligned mode (edge-aligned mode). The direction of the counter is specified by the DIR bit.
TIMER_COUNTER_COUNTER_DOWN	Center-aligned and counting down assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMEx_CHCTL0register). Only when the counter is counting down, compare interrupt flag of channels can be set.
TIMER_COUNTER_COUNTER_UP	Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMEx_CHCTL0register). Only when the counter is counting up, compare interrupt flag of channels can be set.
TIMER_COUNTER_COUNTER	Center-aligned and counting up/down assert mode. The counter counts

<i>NTER_BOTH</i>	under center-aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0 register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set.
<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 3-787. Function timer\_counter\_up\_direction**

<b>Function name</b>	timer_counter_up_direction
<b>Function prototype</b>	void timer_counter_up_direction(uint32_t timer_periph);
<b>Function descriptions</b>	set TIMER counter up direction
<b>Precondition</b>	set TIMER counter no center-aligned mode (edge-aligned mode)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set TIMER0 counter up direction */
timer_counter_up_direction (TIMER0);
```

## timer\_counter\_down\_direction

The description of timer\_counter\_down\_direction is shown as below:

**Table 3-788. Function timer\_counter\_down\_direction**

<b>Function name</b>	timer_counter_down_direction
<b>Function prototype</b>	void timer_counter_down_direction(uint32_t timer_periph);
<b>Function descriptions</b>	set TIMER counter down direction
<b>Precondition</b>	set TIMER counter no center-aligned mode (edge-aligned mode)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7..13)</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 3-789. Function timer\_prescaler\_config**

<b>Function name</b>	timer_prescaler_config
<b>Function prototype</b>	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t pscreload);
<b>Function descriptions</b>	configure TIMER prescaler
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>prescaler</b>	prescaler value (0~65535)
<b>Input parameter{in}</b>	
<b>pscreload</b>	prescaler reload mode
<i>TIMER_PSC_RELOAD_NOW</i>	the prescaler is loaded right now
<i>TIMER_PSC_RELOAD_UPDATE</i>	the prescaler is loaded at the next update event
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 prescaler */
```

```
timer_prescaler_config (TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

### timer\_repetition\_value\_config

The description of timer\_repetition\_value\_config is shown as below:

**Table 3-790. Function timer\_repetition\_value\_config**

<b>Function name</b>	timer_repetition_value_config
<b>Function prototype</b>	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
<b>Function descriptions</b>	configure TIMER repetition register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	

<b>repetition</b>	the counter repetition value (0~255)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 repetition register value */
```

```
timer_repetition_value_config (TIMER0, 98);
```

### timer\_autoreload\_value\_config

The description of timer\_autoreload\_value\_config is shown as below:

**Table 3-791. Function timer\_autoreload\_value\_config**

<b>Function name</b>	timer_autoreload_value_config
<b>Function prototype</b>	void timer_autoreload_value_config(uint32_t timer_periph, uint32_t autoreload);
<b>Function descriptions</b>	configure TIMER autoreload register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>autoreload</b>	the counter auto-reload value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER autoreload register value */
```

```
timer_autoreload_value_config (TIMER0, 3000);
```



## timer\_counter\_value\_config

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

**Table 3-792. 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..13)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>counter</b>	the counter value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 counter register value */
```

```
timer_counter_value_config (TIMER0);
```

## timer\_counter\_read

The description of timer\_counter\_read is shown as below:

**Table 3-793. Function timer\_counter\_read**

<b>Function name</b>	timer_counter_read
<b>Function prototype</b>	uint32_t timer_counter_read(uint32_t timer_periph);
<b>Function descriptions</b>	read TIMER counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	counter value

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 3-794. Function timer\_prescaler\_read**

<b>Function name</b>	timer_prescaler_read
<b>Function prototype</b>	uint16_t timer_prescaler_read(uint32_t timer_periph);
<b>Function descriptions</b>	read TIMER prescaler value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	prescaler register value (0x0000~0xFFFF)

Example:

```
/* read TIMER0 prescaler value */

uint16_t i = 0;

i = timer_prescaler_read (TIMER0);
```

## timer\_single\_pulse\_mode\_config

The description of timer\_single\_pulse\_mode\_config is shown as below:

**Table 3-795. Function timer\_single\_pulse\_mode\_config**

<b>Function name</b>	timer_single_pulse_mode_config
<b>Function prototype</b>	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
<b>Function descriptions</b>	configure TIMER single pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..8,11)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>spmode</b>	pulse mode
<i>TIMER_SP_MODE_SINGLE</i>	single pulse mode
<i>TIMER_SP_MODE_REPETITIVE</i>	repetitive pulse mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 single pulse mode */
```

```
timer_single_pulse_mode_config (TIMER0, TIMER_SP_MODE_SINGLE);
```

## timer\_update\_source\_config

The description of timer\_update\_source\_config is shown as below:

**Table 3-796. Function timer\_update\_source\_config**

<b>Function name</b>	timer_update_source_config
<b>Function prototype</b>	void timer_update_source_config(uint32_t timer_periph, uint32_t update);

<b>Function descriptions</b>	configure TIMER update source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>update</b>	update source
<i>TIMER_UPDATE_SRC_GLOBAL</i>	Any of the following events generate an update interrupt or DMA request: <ul style="list-style-type: none"> <li>- The UPG bit is set</li> <li>- The counter generates an overflow or underflow event</li> <li>- The slave mode controller generates an update event</li> </ul>
<i>TIMER_UPDATE_SRC_REGULAR</i>	Only counter overflow/underflow generates an update interrupt or DMA request.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER update only by counter overflow/underflow */
```

```
timer_update_source_config (TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

### timer\_dma\_enable

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

**Table 3-797. Function timer\_dma\_enable**

<b>Function name</b>	timer_dma_enable
<b>Function prototype</b>	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
<b>Function descriptions</b>	enable the TIMER DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma</b>	timer DMA source enable
<i>TIMER_DMA_UPD</i>	update DMA enable, <i>TIMERx</i> (x=0..7)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable, <i>TIMERx</i> (x=0..4,7)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable, <i>TIMERx</i> (x=0..4,7)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable, <i>TIMERx</i> (x=0..4,7)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable, <i>TIMERx</i> (x=0..4,7)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable, <i>TIMERx</i> (x=0,7)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable, <i>TIMERx</i> (x=0..4,7)
<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 3-798. Function timer\_dma\_disable**

<b>Function name</b>	timer_dma_disable
<b>Function prototype</b>	void timer_dma_disable (uint32_t timer_periph, uint16_t dma);
<b>Function descriptions</b>	disable the TIMER DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma</b>	timer DMA source disable
<i>TIMER_DMA_UPD</i>	update DMA disable, <i>TIMERx</i> (x=0..7)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA disable, <i>TIMERx</i> (x=0..4,7)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA disable, <i>TIMERx</i> (x=0..4,7)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA disable, <i>TIMERx</i> (x=0..4,7)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA disable, <i>TIMERx</i> (x=0..4,7)
<i>TIMER_DMA_CMTD</i>	commutation DMA request disable, <i>TIMERx</i> (x=0,7)
<i>TIMER_DMA_TRGD</i>	trigger DMA disable, <i>TIMERx</i> (x=0..4,7)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the TIMER0 update DMA */
```

```
timer_dma_disable (TIMER0, TIMER_DMA_UPD);
```

### timer\_channel\_dma\_request\_source\_select

The description of timer\_channel\_dma\_request\_source\_select is shown as below:

**Table 3-799. Function timer\_channel\_dma\_request\_source\_select**

<b>Function name</b>	timer_channel_dma_request_source_select
<b>Function prototype</b>	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
<b>Function descriptions</b>	channel DMA request source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection

Input parameter{in}	
<b>dma_request</b>	channel DMA request source selection
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	DMA request of channel n is sent when channel y event occurs
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	DMA request of channel n is sent when update event occurs
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* TIMER0 channel DMA request of channel n is sent when channel y event occurs */
```

```
timer_channel_dma_request_source_select(TIMER0,  
TIMER_DMAREQUEST_CHANNELEVENT);
```

### timer\_dma\_transfer\_config

The description of timer\_dma\_transfer\_config is shown as below:

**Table 3-800. 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>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>dma_baseaddr</b>	DMA transfer access start address
<i>TIMER_DMACFG_DMA_TA_CTL0</i>	DMA transfer address is TIMER_CTL0, TIMERx(x=0..4,7)

<i>TIMER_DMACFG_DMA TA_CTL1</i>	DMA transfer address is TIMER_CTL1, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_SMCFG</i>	DMA transfer address is TIMER_SMCFG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_DMAINTEN</i>	DMA transfer address is TIMER_DMAINTEN, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_INTF</i>	DMA transfer address is TIMER_INTF, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_SWEVG</i>	DMA transfer address is TIMER_SWEVG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CHCTL0</i>	DMA transfer address is TIMER_CHCTL0, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CHCTL1</i>	DMA transfer address is TIMER_CHCTL1, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CHCTL2</i>	DMA transfer address is TIMER_CHCTL2, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CNT</i>	DMA transfer address is TIMER_CNT, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_PSC</i>	DMA transfer address is TIMER_PSC, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CAR</i>	MA transfer address is TIMER_CAR, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CREP</i>	DMA transfer address is TIMER_CREP, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMA TA_CH0CV</i>	DMA transfer address is TIMER_CH0CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CH1CV</i>	DMA transfer address is TIMER_CH1CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CH2CV</i>	DMA transfer address is TIMER_CH2CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CH3CV</i>	DMA transfer address is TIMER_CH3CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CCHP</i>	DMA transfer address is TIMER_CCHP, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMA</i>	DMA transfer address is TIMER_DMACFG, TIMERx(x=0..4,7)



<i>TA_DMACFG</i>	
<i>TIMER_DMACFG_DMA</i> <i>TA_DMATB</i>	DMA transfer address is TIMER_DMATB, TIMERx(x=0..4,7)
<b>Input parameter{in}</b>	
<b>dma_lenth</b>	DMA transfer count
<i>TIMER_DMACFG_DMA</i> <i>TC_xTRANSFER</i>	x=1..18, DMA transfer x time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the TIMER0 DMA transfer */
```

```
timer_dma_transfer_config(TIMER0, TIMER_DMACFG_DMATA_CTL0,  
TIMER_DMACFG_DMATC_5TRANSFER);
```

### timer\_event\_software\_generate

The description of timer\_event\_software\_generate is shown as below:

**Table 3-801. Function timer\_event\_software\_generate**

<b>Function name</b>	timer_event_software_generate
<b>Function prototype</b>	void timer_event_software_generate(uint32_t timer_periph, uint16_t event);
<b>Function descriptions</b>	software generate events
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>event</b>	the timer software event generation sources
<i>TIMER_EVENT_SRC_U</i>	update event, TIMERx(x=0..13)

<i>PG</i>	
<i>TIMER_EVENT_SRC_C</i> <i>H0G</i>	channel 0 capture or compare event generation,TIMERx(x=0..4,7..13)
<i>TIMER_EVENT_SRC_C</i> <i>H1G</i>	channel 1 capture or compare event generation,TIMERx(x=0..4,7,8,11)
<i>TIMER_EVENT_SRC_C</i> <i>H2G</i>	channel 2 capture or compare event generation,TIMERx(x=0..4,7)
<i>TIMER_EVENT_SRC_C</i> <i>H3G</i>	channel 3 capture or compare event generation,TIMERx(x=0..4,7)
<i>TIMER_EVENT_SRC_C</i> <i>MTG</i>	channel commutation event generation,TIMERx(x=0,7)
<i>TIMER_EVENT_SRC_T</i> <i>RGG</i>	trigger event generation,TIMERx(x=0..4,7,8,11)
<i>TIMER_EVENT_SRC_B</i> <i>RKG</i>	break event generation,TIMERx(x=0,7)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* software generate update event*/
```

```
timer_event_software_generate (TIMER0, TIMER_EVENT_SRC_UPG);
```

### timer\_break\_struct\_para\_init

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

**Table 3-802. Function timer\_break\_struct\_para\_init**

<b>Function name</b>	timer_break_struct_para_init
<b>Function prototype</b>	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
<b>Function descriptions</b>	initialize the parameters of TIMER break parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>breakpara</b>	TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-774. 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 3-803. 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)</i>	TIMER peripheral selection
Input parameter{in}	
<b>breakpara</b>	TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-774. Structure timer break parameter struct.</a>
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure TIMER0 break function */

timer_break_parameter_struct timer_breakpara;

timer_breakpara.runoffstate      = TIMER_ROS_STATE_DISABLE;
timer_breakpara.ideloffstate     = TIMER_IOS_STATE_DISABLE ;
timer_breakpara.deadtime         = 255;
timer_breakpara.breakpolarity    = TIMER_BREAK_POLARITY_LOW;
timer_breakpara.outputautostate  = TIMER_OUTAUTO_ENABLE;
timer_breakpara.protectmode       = TIMER_CCHP_PROT_0;
timer_breakpara.breakstate       = TIMER_BREAK_ENABLE;

timer_break_config(TIMER0, &timer_breakpara);
```

### timer\_break\_enable

The description of timer\_break\_enable is shown as below:

**Table 3-804. Function timer\_break\_enable**

<b>Function name</b>	timer_break_enable
<b>Function prototype</b>	void timer_break_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable TIMER break function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 break function*/
```

```
timer_break_enable (TIMER0);
```

### timer\_break\_disable

The description of timer\_break\_disable is shown as below:

**Table 3-805. Function timer\_break\_disable**

<b>Function name</b>	timer_break_disable
<b>Function prototype</b>	void timer_break_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER break function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 break function*/
```

```
timer_break_disable (TIMER0);
```

### timer\_automatic\_output\_enable

The description of timer\_automatic\_output\_enable is shown as below:

**Table 3-806. Function timer\_automatic\_output\_enable**

<b>Function name</b>	timer_automatic_output_enable
<b>Function prototype</b>	void timer_automatic_output_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable TIMER output automatic function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMER<sub>x</sub>(x=0,7)</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 3-807. 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 TIMER <sub>x</sub> _CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMER<sub>x</sub>(x=0,7)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 output automatic function */
```

timer\_automatic\_output\_disable (TIMER0);

### timer\_primary\_output\_config

The description of timer\_primary\_output\_config is shown as below:

**Table 3-808. Function timer\_primary\_output\_config**

<b>Function name</b>	timer_primary_output_config
<b>Function prototype</b>	void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue);
<b>Function descriptions</b>	configure TIMER primary output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TIMER0 primary output function */
```

```
timer_primary_output_config (TIMER0, ENABLE);
```

### timer\_channel\_control\_shadow\_config

The description of timer\_channel\_control\_shadow\_config is shown as below:

**Table 3-809. Function timer\_channel\_control\_shadow\_config**

<b>Function name</b>	timer_channel_control_shadow_config
<b>Function prototype</b>	void timer_channel_control_shadow_config(uint32_t timer_periph,

	ControlStatus newvalue);
<b>Function descriptions</b>	channel commutation control shadow register enable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMER<sub>x</sub></i> ( <i>x</i> =0,7)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* channel capture/compare control shadow register enable */
```

```
timer_channel_control_shadow_config (TIMER0, ENABLE);
```

### timer\_channel\_control\_shadow\_update\_config

The description of timer\_channel\_control\_shadow\_update\_config is shown as below:

**Table 3-810. Function timer\_channel\_control\_shadow\_update\_config**

<b>Function name</b>	timer_channel_control_shadow_update_config
<b>Function prototype</b>	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint8_t ccuctl);
<b>Function descriptions</b>	configure commutation control shadow register update control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral



<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>ccuctl</b>	channel control shadow register update control
<i>TIMER_UPDATECTL_CCU</i>	the shadow registers update by when CMTG bit is set
<i>TIMER_UPDATECTL_CUTRI</i>	the shadow registers update by when CMTG bit is set or an rising edge of TRGI occurs
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel control shadow register update when CMTG bit is set */
timer_channel_control_shadow_update_config (TIMER0, TIMER_UPDATECTL_CCU);
```

### timer\_channel\_output\_struct\_para\_init

The description of timer\_channel\_output\_struct\_para\_init is shown as below:

**Table 3-811. Function timer\_channel\_output\_struct\_para\_init**

<b>Function name</b>	timer_channel_output_struct_para_init
<b>Function prototype</b>	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpa);
<b>Function descriptions</b>	initialize the parameters of TIMER channel output parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ocpa</b>	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-775. 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 3-812. Function timer\_channel\_output\_config**

<b>Function name</b>	timer_channel_output_config
<b>Function prototype</b>	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpa);
<b>Function descriptions</b>	configure TIMER channel output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>ocpara</b>	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-775. 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 3-813. Function timer\_channel\_output\_mode\_config**

Function name	timer_channel_output_mode_config
Function prototype	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);
Function descriptions	configure TIMER channel output compare mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMEx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMEx(x=0..4,7..13))
TIMER_CH_1	TIMER channel 1 (TIMEx(x=0..4,7,8,11))
TIMER_CH_2	TIMER channel 2 (TIMEx(x=0..4,7))
TIMER_CH_3	IMER channel 3 (TIMEx(x=0..4,7))

Input parameter{in}	
<b>ocmode</b>	channel output compare mode
<i>TIMER_OC_MODE_TIMING</i>	timing mode
<i>TIMER_OC_MODE_ACTIVE</i>	set the channel output
<i>TIMER_OC_MODE_INACTIVE</i>	clear the channel output
<i>TIMER_OC_MODE_TOGGLE</i>	toggle on match
<i>TIMER_OC_MODE_LOW</i>	force low mode
<i>TIMER_OC_MODE_HIGH</i>	force high mode
<i>TIMER_OC_MODE_PWM0</i>	PWM mode 0
<i>TIMER_OC_MODE_PWM1</i>	PWM mode 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER0, TIMER_CH_0, TIMER_OC_MODE_PWM0);
```

### timer\_channel\_output\_pulse\_value\_config

The description of timer\_channel\_output\_pulse\_value\_config is shown as below:

**Table 3-814. Function timer\_channel\_output\_pulse\_value\_config**

<b>Function name</b>	timer_channel_output_pulse_value_config
<b>Function prototype</b>	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);
<b>Function descriptions</b>	configure TIMER channel output pulse value

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>pulse</b>	channel output pulse value (0~65535)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output pulse value */
```

```
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

### timer\_channel\_output\_shadow\_config

The description of timer\_channel\_output\_shadow\_config is shown as below:

**Table 3-815. Function timer\_channel\_output\_shadow\_config**

<b>Function name</b>	timer_channel_output_shadow_config
<b>Function prototype</b>	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
<b>Function descriptions</b>	configure TIMER channel output shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
<b>ocshadow</b>	channel output shadow state
<i>TIMER_OC_SHADOW_ENABLE</i>	channel output shadow state enable
<i>TIMER_OC_SHADOW_DISABLE</i>	channel output shadow state disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*configure TIMER0 channel 0 output shadow function */
timer_channel_output_shadow_config (TIMER0, TIMER_CH_0,
TIMER_OC_SHADOW_ENABLE);
```

### timer\_channel\_output\_fast\_config

The description of timer\_channel\_output\_fast\_config is shown as below:

**Table 3-816. Function timer\_channel\_output\_fast\_config**

<b>Function name</b>	timer_channel_output_fast_config
<b>Function prototype</b>	void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
<b>Function descriptions</b>	configure TIMER channel output fast function

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>ocfast</b>	channel output fast function
<i>TIMER_OC_FAST_ENABLE</i>	channel output fast function enable
<i>TIMER_OC_FAST_DISABLE</i>	channel output fast function disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output fast function */
```

```
timer_channel_output_fast_config (TIMER0, TIMER_CH_0, TIMER_OC_FAST_ENABLE);
```

### timer\_channel\_output\_clear\_config

The description of timer\_channel\_output\_clear\_config is shown as below:

**Table 3-817. Function timer\_channel\_output\_clear\_config**

<b>Function name</b>	timer_channel_output_clear_config
<b>Function prototype</b>	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t

	channel, uint16_t occlear);
<b>Function descriptions</b>	configure TIMER channel output clear function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER periphera
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>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 3-818. Function timer\_channel\_output\_polarity\_config

<b>Function name</b>	timer_channel_output_polarity_config
<b>Function prototype</b>	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
<b>Function descriptions</b>	configure TIMER channel output polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>ocpolarity</b>	channel output polarity
<i>TIMER_OC_POLARITY_HIGH</i>	channel output polarity is high
<i>TIMER_OC_POLARITY_LOW</i>	channel output polarity is low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 output polarity */
```

```
timer_channel_output_polarity_config (TIMER0, TIMER_CH_0,  
TIMER_OC_POLARITY_HIGH);
```

## timer\_channel\_complementary\_output\_polarity\_config

The description of timer\_channel\_complementary\_output\_polarity\_config is shown as below:

**Table 3-819. Function timer\_channel\_complementary\_output\_polarity\_config**

<b>Function name</b>	timer_channel_complementary_output_polarity_config
<b>Function prototype</b>	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnpolarity);
<b>Function descriptions</b>	configure TIMER channel complementary output polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<i>TIMER_CH_2</i>	TIMER channel 2
<b>Input parameter{in}</b>	
<b>ocpolarity</b>	channel complementary output polarity
<i>TIMER_OCN_POLARITY_HIGH</i>	channel complementary output polarity is high
<i>TIMER_OCN_POLARITY_LOW</i>	channel complementary output polarity is low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output polarity */
timer_channel_complementary_output_polarity_config (TIMER0, TIMER_CH_0,
```

TIMER\_OCN\_POLARITY\_HIGH);

### timer\_channel\_output\_state\_config

The description of timer\_channel\_output\_state\_config is shown as below:

**Table 3-820. Function timer\_channel\_output\_state\_config**

<b>Function name</b>	timer_channel_output_state_config
<b>Function prototype</b>	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
<b>Function descriptions</b>	configure TIMER channel enable state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>state</b>	TIMER channel enable state
<i>TIMER_CCX_ENABLE</i>	channel enable
<i>TIMER_CCX_DISABLE</i>	channel disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 enable state */
```

timer\_channel\_output\_state\_config (TIMER0, TIMER\_CH\_0, TIMER\_CCX\_ENABLE);

### timer\_channel\_complementary\_output\_state\_config

The description of timer\_channel\_complementary\_output\_state\_config is shown as below:

**Table 3-821. Function timer\_channel\_complementary\_output\_state\_config**

<b>Function name</b>	timer_channel_complementary_output_state_config
<b>Function prototype</b>	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
<b>Function descriptions</b>	configure TIMER channel complementary output enable state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<i>TIMER_CH_2</i>	TIMER channel 2
<b>Input parameter{in}</b>	
<b>state</b>	TIMER channel complementary output enable state
<i>TIMER_CCXN_ENABLE</i>	channel complementary enable
<i>TIMER_CCXN_DISABLE</i>	channel complementary disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output enable state */
```

```
timer_channel_complementary_output_state_config (TIMER0, TIMER_CH_0,
```

TIMER\_CCXN\_ENABLE);

### timer\_channel\_input\_struct\_para\_init

The description of timer\_channel\_input\_struct\_para\_init is shown as below:

**Table 3-822. Function timer\_channel\_input\_struct\_para\_init**

<b>Function name</b>	timer_channel_input_struct_para_init
<b>Function prototype</b>	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
<b>Function descriptions</b>	initialize the parameters of TIMER channel input parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
icpara	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-776. Structure timer ic parameter struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize TIMER channel input parameter struct with a default value */
timer_ic_parameter_struct timer_icinitpara;
timer_channel_input_struct_para_init(timer_icinitpara);
```

### timer\_input\_capture\_config

The description of timer\_input\_capture\_config is shown as below:

**Table 3-823. Function timer\_input\_capture\_config**

<b>Function name</b>	timer_input_capture_config
<b>Function prototype</b>	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
<b>Function descriptions</b>	configure TIMER input capture parameter
<b>Precondition</b>	-

<b>The called functions</b>	timer_channel_input_capture_prescaler_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>icpara</b>	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-776. Structure timer ic parameter struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* configure TIMER0 input capture parameter */
timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;

timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);

```

### timer\_channel\_input\_capture\_prescaler\_config

The description of timer\_channel\_input\_capture\_prescaler\_config is shown as below:

**Table 3-824. Function timer\_channel\_input\_capture\_prescaler\_config**

<b>Function name</b>	timer_channel_input_capture_prescaler_config
----------------------	----------------------------------------------

<b>Function prototype</b>	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);
<b>Function descriptions</b>	configure TIMER channel input capture prescaler value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Input parameter{in}</b>	
<b>prescaler</b>	channel input capture prescaler value
<i>TIMER_IC_PSC_DIV1</i>	no prescaler
<i>TIMER_IC_PSC_DIV2</i>	divided by 2
<i>TIMER_IC_PSC_DIV4</i>	divided by 4
<i>TIMER_IC_PSC_DIV8</i>	divided by 8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 input capture prescaler value */
```

```
timer_channel_input_capture_prescaler_config (TIMER0, TIMER_CH_0,  
TIMER_IC_PSC_DIV2);
```

## timer\_channel\_capture\_value\_register\_read

The description of timer\_channel\_capture\_value\_register\_read is shown as below:

**Table 3-825. Function timer\_channel\_capture\_value\_register\_read**

<b>Function name</b>	timer_channel_capture_value_register_read
<b>Function prototype</b>	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
<b>Function descriptions</b>	read TIMER channel capture compare register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	IMER channel 3 (TIMERx(x=0..4,7))
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	channel capture compare register value (0x0000~0xFFFF)

Example:

```
/* read TIMER0 channel 0 capture compare register value */
uint32_t ch0_value = 0;
ch0_value = timer_channel_capture_value_register_read (TIMER0, TIMER_CH_0);
```

## timer\_input\_pwm\_capture\_config

The description of timer\_input\_pwm\_capture\_config is shown as below:



Table 3-826. Function timer\_input\_pwm\_capture\_config

Function name	timer_input_pwm_capture_config
Function prototype	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
Function descriptions	configure TIMER input pwm capture function
Precondition	-
The called functions	timer_channel_input_capture_prescaler_config
Input parameter{in}	
timer_periph	TIMER peripheral
TIMER <sub>x</sub> ( <i>x</i> =0..4,7,8,11)	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0
TIMER_CH_1	TIMER channel 1
Input parameter{in}	
icpwm	TIMER channel input pwm parameter struct, the structure members can refer to <a href="#">Table 3-776. Structure timer_ic_parameter_struct</a> .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 input pwm capture parameter */
timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;

timer_input_pwm_capture_config (TIMER0, TIMER_CH_0, &timer_icinitpara);

```

## timer\_hall\_mode\_config

The description of timer\_hall\_mode\_config is shown as below:

**Table 3-827. Function timer\_hall\_mode\_config**

<b>Function name</b>	timer_hall_mode_config
<b>Function prototype</b>	void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode);
<b>Function descriptions</b>	configure TIMER hall sensor mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>hallmode</b>	TIMER hall sensor mode state
<i>TIMER_HALLINTERFACE_CE_ENABLE</i>	TIMER hall sensor mode enable
<i>TIMER_HALLINTERFACE_CE_DISABLE</i>	TIMER hall sensor mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 hall sensor mode */
```

```
timer_hall_mode_config (TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

## timer\_input\_trigger\_source\_select

The description of timer\_input\_trigger\_source\_select is shown as below:

**Table 3-828. Function timer\_input\_trigger\_source\_select**

<b>Function name</b>	timer_input_trigger_source_select
<b>Function prototype</b>	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);

<b>Function descriptions</b>	select TIMER input trigger source
<b>Precondition</b>	SMC[2:0] = 000
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI0</i>	Internal trigger input 0 (ITI0, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI1</i>	Internal trigger input 0 (ITI1, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI2</i>	Internal trigger input 0 (ITI2, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI3</i>	Internal trigger input 0 (ITI3, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CIO edge flag (CIOF_ED, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output (CIOFE0, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_C1FE1</i>	channel 1 input Filtered output (C1FE1, TIMERx(x=0..4,7,8,11) )
<i>TIMER_SMCFG_TRGS</i> <i>EL_ETIFP</i>	External trigger input filter output(ETIFP, TIMERx(x=0..4,7) )
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER0 input trigger source */
```

```
timer_input_trigger_source_select (TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

## timer\_master\_output\_trigger\_source\_select

The description of timer\_master\_output\_trigger\_source\_select is shown as below:

**Table 3-829. Function timer\_master\_output\_trigger\_source\_select**

<b>Function name</b>	timer_master_output_trigger_source_select
<b>Function prototype</b>	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
<b>Function descriptions</b>	select TIMER master mode output trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>outrigger</b>	master mode control
<i>TIMER_TRI_OUT_SRC_RESET</i>	Reset. When the UPG bit in the <i>TIMERx_SWEVG</i> register is set or a reset is generated by the slave mode controller, a TRGO pulse occurs. And in the latter case, the signal on TRGO is delayed compared to the actual reset
<i>TIMER_TRI_OUT_SRC_ENABLE</i>	Enable. This mode is useful to start several timers at the same time or to control a window in which a slave timer is enabled. In this mode the master mode controller selects the counter enable signal as TRGO. The counter enable signal is set when CEN control bit is set or the trigger input in pause mode is high. There is a delay between the trigger input in pause mode and the TRGO output, except if the master-slave mode is selected.
<i>TIMER_TRI_OUT_SRC_UPDATE</i>	Update. In this mode the master mode controller selects the update event as TRGO.
<i>TIMER_TRI_OUT_SRC_CC0</i>	Capture/compare pulse. In this mode the master mode controller generates a TRGO pulse when a capture or a compare match occurred in channel 0.
<i>TIMER_TRI_OUT_SRC_O0CPRE</i>	Compare. In this mode the master mode controller selects the O0CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O1CPRE</i>	Compare. In this mode the master mode controller selects the O1CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O2CPRE</i>	Compare. In this mode the master mode controller selects the O2CPRE signal is used as TRGO.

<i>TIMER_TRI_OUT_SRC</i> <i>_O3CPRE</i>	Compare. In this mode the master mode controller selects the O3CPRE signal is used as TRGO.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER0 master mode output trigger source */
timer_master_output_trigger_source_select (TIMER0, TIMER_TRI_OUT_SRC_RESET);
```

### timer\_slave\_mode\_select

The description of timer\_slave\_mode\_select is shown as below:

**Table 3-830. Function timer\_slave\_mode\_select**

<b>Function name</b>	timer_slave_mode_select
<b>Function prototype</b>	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
<b>Function descriptions</b>	select TIMER slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>slavemode</b>	slave mode
<i>TIMER_SLAVE_MODE_DISABLE</i>	slave mode disable
<i>TIMER_QUAD_DECODER_MODE0</i>	quadrature decoder mode 0
<i>TIMER_QUAD_DECODER_MODE1</i>	quadrature decoder mode 1
<i>TIMER_QUAD_DECODER_MODE2</i>	quadrature decoder mode 2

<i>TIMER_SLAVE_MODE_RESTART</i>	restart mode
<i>TIMER_SLAVE_MODE_PAUSE</i>	pause mode
<i>TIMER_SLAVE_MODE_EVENT</i>	event mode
<i>TIMER_SLAVE_MODE_EXTERNAL0</i>	external clock mode 0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER0 slave mode */
```

```
timer_slave_mode_select (TIMER0, TIMER_QUAD_DECODER_MODE0);
```

### timer\_master\_slave\_mode\_config

The description of timer\_master\_slave\_mode\_config is shown as below:

**Table 3-831. Function timer\_master\_slave\_mode\_config**

<b>Function name</b>	timer_master_slave_mode_config
<b>Function prototype</b>	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);
<b>Function descriptions</b>	configure TIMER master slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>masterslave</b>	master slave mode state
<i>TIMER_MASTER_SLAV</i>	master slave mode enable

<i>E_MODE_ENABLE</i>	
<i>TIMER_MASTER_SLAVE</i> <i>E_MODE_DISABLE</i>	master slave mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 master slave mode */
```

```
timer_master_slave_mode_config (TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

### timer\_external\_trigger\_config

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

**Table 3-832. Function timer\_external\_trigger\_config**

<b>Function name</b>	timer_external_trigger_config
<b>Function prototype</b>	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER external trigger input
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4

<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
<b>Input parameter{in}</b>	
<b>expolarity</b>	external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 external trigger input */
```

```
timer_external_trigger_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 10);
```

### timer\_quadrature\_decoder\_mode\_config

The description of timer\_quadrature\_decoder\_mode\_config is shown as below:

**Table 3-833. 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 decompode, 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>TIMER<sub>x</sub></i> ( <i>x</i> =0..4,7)	TIMER peripheral selection
<b>Input parameter{in}</b>	



<b>decomode</b>	quadrature decoder mode
<i>TIMER_QUAD_DECODER_MODE0</i>	counter counts on CI0FE0 edge depending on CI1FE1 level
<i>TIMER_QUAD_DECODER_MODE1</i>	counter counts on CI1FE1 edge depending on CI0FE0 level
<i>TIMER_QUAD_DECODER_MODE2</i>	counter counts on both CI0FE0 and CI1FE1 edges depending on the level of the other input
<b>Input parameter{in}</b>	
<b>ic0polarity</b>	IC0 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<b>Input parameter{in}</b>	
<b>ic1polarity</b>	IC1 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config (TIMER0, TIMER_QUAD_DECODER_MODE0,
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

### timer\_internal\_clock\_config

The description of timer\_internal\_clock\_config is shown as below:

**Table 3-834. Function timer\_internal\_clock\_config**

<b>Function name</b>	timer_internal_clock_config
----------------------	-----------------------------

<b>Function prototype</b>	void timer_internal_clock_config(uint32_t timer_periph);
<b>Function descriptions</b>	configure TIMER internal clock mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 internal clock mode */
```

```
timer_internal_clock_config (TIMER0);
```

### timer\_internal\_trigger\_as\_external\_clock\_config

The description of timer\_internal\_trigger\_as\_external\_clock\_config is shown as below:

**Table 3-835. Function timer\_internal\_trigger\_as\_external\_clock\_config**

<b>Function name</b>	timer_internal_trigger_as_external_clock_config
<b>Function prototype</b>	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
<b>Function descriptions</b>	configure TIMER the internal trigger as external clock input
<b>Precondition</b>	-
<b>The called functions</b>	timer_input_trigger_source_select
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS</i>	Internal trigger input 0 (ITI0)

<i>EL_ITI0</i>	
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI1</i>	Internal trigger input 0 (ITI1)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI2</i>	Internal trigger input 0 (ITI2)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI3</i>	Internal trigger input 0 (ITI3)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 the internal trigger ITI0 as external clock input */
```

```
timer_internal_trigger_as_external_clock_config (TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

### **timer\_external\_trigger\_as\_external\_clock\_config**

The description of timer\_external\_trigger\_as\_external\_clock\_config is shown as below:

**Table 3-836. Function timer\_external\_trigger\_as\_external\_clock\_config**

<b>Function name</b>	timer_external_trigger_as_external_clock_config
<b>Function prototype</b>	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external trigger as external clock input
<b>Precondition</b>	-
<b>The called functions</b>	timer_input_trigger_source_select
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMER<sub>x</sub></i> ( <i>x</i> =0..4,7,8,11)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extrigger</b>	external trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CIO edge flag (CIOF_ED)

<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output (CI0FE0)
<i>TIMER_SMCFG_TRGS</i> <i>EL_C11FE1</i>	channel 1 input Filtered output (CI1FE1)
<b>Input parameter{in}</b>	
<b>expolarity</b>	external trigger polarity
<i>TIMER_IC_POLARITY_RISING</i>	active high or rising edge active
<i>TIMER_IC_POLARITY_FALLING</i>	active low or falling edge active
<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 3-837. Function timer\_external\_clock\_mode0\_config**

<b>Function name</b>	timer_external_clock_mode0_config
<b>Function prototype</b>	void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external clock mode0
<b>Precondition</b>	-
<b>The called functions</b>	timer_external_trigger_config
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMER<sub>x</sub></i> ( <i>x</i> =0..4,7,8,11)	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
<b>Input parameter{in}</b>	
<b>expolarity</b>	ETI external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	ETI external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

### timer\_external\_clock\_mode1\_config

The description of timer\_external\_clock\_mode1\_config is shown as below:

**Table 3-838. 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>TIMER<sub>x</sub></i> ( <i>x</i> =0..4,7)	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 3-839. Function timer\_external\_clock\_mode1\_disable**

<b>Function name</b>	timer_external_clock_mode1_disable
<b>Function prototype</b>	void timer_external_clock_mode1_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER the external clock mode1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMER<sub>x</sub>(x=0..4,7)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 the external clock mode1 */
timer_external_clock_mode1_disable (TIMER0);
```

### timer\_channel\_remap\_config

The description of timer\_channel\_remap\_config is shown as below:

**Table 3-840. Function timer\_channel\_remap\_config**

<b>Function name</b>	timer_channel_remap_config
<b>Function prototype</b>	void timer_channel_remap_config(uint32_t timer_periph, uint32_t remap);
<b>Function descriptions</b>	configure TIMER channel remap function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=1,4,10)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>remap</b>	TIMER remap selection
<i>TIMER1_ITI1_RMP_TIMER7_TRGO</i>	timer1 internal trigger input1 remap to TIMER7_TRGO
<i>TIMER1_ITI1_RMP_ETH_ETHERNET_PTP</i>	timer1 internal trigger input1 remap to ethernet PTP
<i>TIMER1_ITI1_RMP_USB_FS_SOF</i>	timer1 internal trigger input1 remap to USB FS SOF
<i>TIMER1_ITI1_RMP_USB_HS_SOF</i>	timer1 internal trigger input1 remap to USB HS SOF
<i>TIMER4_CI3_RMP_GPIO</i>	timer4 channel 3 input remap to GPIO pin
<i>TIMER4_CI3_RMP_IRC32K</i>	timer4 channel 3 input remap to IRC32K
<i>TIMER4_CI3_RMP_LXTAL</i>	timer4 channel 3 input remap to LXTAL
<i>TIMER4_CI3_RMP_RTC_WAKEUP_INT</i>	timer4 channel 3 input remap to RTC wakeup interrupt
<i>TIMER10_ITI1_RMP_GPIO</i>	internal trigger input1 remap based on GPIO setting
<i>TIMER10_ITI1_RMP_RTC_HXTAL_DIV</i>	timer10 internal trigger input1 remap HXTAL_DIV(clock used for RTC which is HXTAL clock divided by RTCDIV bits in RCU_CFG0 register)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure timer1 internal trigger input1 remap to TIMER7_TRGO */
timer_channel_remap_config (TIMER1,TIMER1_ITI1_RMP_TIMER7_TRGO);
```



## timer\_write\_chxval\_register\_config

The description of timer\_write\_chxval\_register\_config is shown as below:

**Table 3-841. Function timer\_write\_chxval\_register\_config**

<b>Function name</b>	timer_write_chxval_register_config
<b>Function prototype</b>	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
<b>Function descriptions</b>	configure TIMER write CHxVAL register selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0..4,7..13)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>ccsel</b>	write CHxVAL register selection
<i>TIMER_CHVSEL_DISABLE</i>	no effect
<i>TIMER_CHVSEL_ENABLE</i>	when write the CHxVAL register, if the write value is same as the CHxVAL value, the write access is ignored
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 write CHxVAL register selection */
timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

## timer\_output\_value\_selection\_config

The description of timer\_output\_value\_selection\_config is shown as below:

**Table 3-842. Function timer\_output\_value\_selection\_config**

<b>Function name</b>	timer_output_value_selection_config
<b>Function prototype</b>	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t

	outsel);
<b>Function descriptions</b>	configure TIMER output value selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx (x=0,7)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>outsel</b>	output value selection
<i>TIMER_OUTSEL_DISABLE</i>	no effect
<i>TIMER_OUTSEL_ENABLE</i>	if POEN and IOS is 0, the output disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER output value selection */
```

```
timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

### timer\_flag\_get

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

**Table 3-843. Function timer\_flag\_get**

<b>Function name</b>	timer_flag_get
<b>Function prototype</b>	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
<b>Function descriptions</b>	get TIMER flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag, <i>TIMERx</i> ( <i>x</i> =0..13)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7..13)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_TRG</i>	trigger flag, <i>TIMERx</i> ( <i>x</i> =0,7,8,11)
<i>TIMER_FLAG_BRK</i>	break flag, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..4,7..11)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..4,7,8,11)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, <i>TIMERx</i> ( <i>x</i> =0..4,7)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TIMER0 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get (TIMER0, TIMER_FLAG_UP);
```

### timer\_flag\_clear

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

**Table 3-844. Function timer\_flag\_clear**

<b>Function name</b>	timer_flag_clear
----------------------	------------------

<b>Function prototype</b>	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
<b>Function descriptions</b>	clear TIMER flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag, TIMERx(x=0..13)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, TIMERx(x=0..4,7..13)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, TIMERx(x=0..4,7,8,11)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, TIMERx(x=0..4,7)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, TIMERx(x=0..4,7)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, TIMERx(x=0,7)
<i>TIMER_FLAG_TRG</i>	trigger flag, TIMERx(x=0,7,8,11)
<i>TIMER_FLAG_BRK</i>	break flag, TIMERx(x=0,7)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, TIMERx(x=0..4,7..11)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, TIMERx(x=0..4,7,8,11)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, TIMERx(x=0..4,7)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, TIMERx(x=0..4,7)
<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 3-845. Function timer\_interrupt\_enable**

<b>Function name</b>	timer_interrupt_enable
<b>Function prototype</b>	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable the TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	timer interrupt enable source
<i>TIMER_INT_UP</i>	update interrupt enable, TIMERx (x=0..13)
<i>TIMER_INT_CH0</i>	channel 0 interrupt enable, TIMERx (x=0..4,7..13)
<i>TIMER_INT_CH1</i>	channel 1 interrupt enable, TIMERx (x=0..4,7,8,11)
<i>TIMER_INT_CH2</i>	channel 2 interrupt enable, TIMERx (x=0..4,7)
<i>TIMER_INT_CH3</i>	channel 3 interrupt enable , TIMERx (x=0..4,7)
<i>TIMER_INT_CMT</i>	commutation interrupt enable, TIMERx (x=0,7)
<i>TIMER_INT_TRG</i>	trigger interrupt enable, TIMERx (x=0..4,7,8,11)
<i>TIMER_INT_BRK</i>	break interrupt enable, TIMERx (x=0,7)
<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 3-846. Function timer\_interrupt\_disable**

<b>Function name</b>	timer_interrupt_disable
<b>Function prototype</b>	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable the TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	timer interrupt disable source
<i>TIMER_INT_UP</i>	update interrupt disable, TIMERx(x=0..13)
<i>TIMER_INT_CH0</i>	channel 0 interrupt disable, TIMERx(x=0..4,7..13)
<i>TIMER_INT_CH1</i>	channel 1 interrupt disable, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_CH2</i>	channel 2 interrupt disable, TIMERx(x=0..4,7)
<i>TIMER_INT_CH3</i>	channel 3 interrupt disable, TIMERx(x=0..4,7)
<i>TIMER_INT_CMT</i>	commutation interrupt disable, TIMERx(x=0,7)
<i>TIMER_INT_TRG</i>	trigger interrupt disable, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_BRK</i>	break interrupt disable, 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 3-847. Function timer\_interrupt\_flag\_get**

<b>Function name</b>	timer_interrupt_flag_get
<b>Function prototype</b>	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	get timer interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, TIMERx(x=0..13)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, TIMERx(x=0..4,7..13)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, TIMERx(x=0..4,7)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, TIMERx(x=0..4,7)
<i>TIMER_INT_FLAG_CMT</i>	channel commutation interrupt flag, TIMERx(x=0,7)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, TIMERx(x=0,7,8,11)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, TIMERx(x=0,7)
<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 (TIMER0, TIMER_INT_FLAG_UP);
```

### timer\_interrupt\_flag\_clear

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

**Table 3-848. Function timer\_interrupt\_flag\_clear**

<b>Function name</b>	timer_interrupt_flag_clear
<b>Function prototype</b>	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	clear TIMER interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, TIMERx(x=0..13)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, TIMERx(x=0..4,7..13)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, TIMERx(x=0..4,7)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, TIMERx(x=0..4,7)
<i>TIMER_INT_FLAG_CMT</i>	channel commutation interrupt flag, TIMERx(x=0,7)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, TIMERx(x=0,7,8,11)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, TIMERx(x=0,7)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER0 update interrupt flag */
```



timer\_interrupt\_flag\_clear (TIMER0, TIMER\_INT\_FLAG\_UP);

## 3.27. TLI

The TLI (TFT-LCD Interface) module handles the synchronous LCD interface and provides pixel data, clock and timing signals for passive LCD display. The TLI registers are listed in chapter [3.27.1](#), the TLI firmware functions are introduced in chapter [3.27.2](#).

### 3.27.1. Descriptions of Peripheral registers

TLI registers are listed in the table shown as below:

**Table 3-849. TLI Registers**

Registers	Descriptions
TLI_SPSZ	TLI synchronous pulse size register
TLI_BPSZ	TLI back-porch size register
TLI_ASZ	TLI active size register
TLI_TSZ	TLI total size register
TLI_CTL	TLI control register
TLI_RL	TLI reload Layer register
TLI_BGC	TLI background color register
TLI_INTEN	TLI interrupt enable register
TLI_INTF	TLI interrupt flag register
TLI_INTC	TLI interrupt flag clear register
TLI_LM	TLI line mark register
TLI_CPPOS	TLI current pixel position register
TLI_STAT	TLI status register
TLI_LxCTL	TLI layer x control register
TLI_LxHPOS	TLI layer x horizontal position parameters register
TLI_LxVPOS	TLI layer x vertical position parameters register
TLI_LxCKEY	TLI layer x color key register
TLI_LxPPF	TLI layer x packeted pixel format register
TLI_LxSA	TLI layer x specified alpha register

Registers	Descriptions
TLI_LxDC	TLI layer x default color register
TLI_LxBLEND	TLI layer x blending register
TLI_LxFBADDR	TLI layer x frame base address register
TLI_LxFLEN	TLI layer x frame line length register
TLI_LxFTLN	TLI layer x frame total line number register
TLI_LxLUT	TLI layer x Look Up Table register

### 3.27.2. Descriptions of Peripheral functions

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

**Table 3-850. TLI firmware function**

Function name	Function description
tli_deinit	deinitialize TLI registers
tli_struct_para_init	initialize the parameters of TLI parameter structure with the default values, it is suggested that call this function after a tli_parameter_struct structure is defined
tli_init	initialize TLI
tli_dither_config	configure TLI dither function
tli_enable	enable TLI
tli_disable	disable TLI
tli_reload_config	configure TLI reload mode
tli_layer_struct_para_init	initialize the parameters of TLI layer structure with the default values, it is suggested that call this function after a tli_layer_parameter_struct structure is defined
tli_layer_init	initialize TLI layer
tli_layer_window_offset_modify	reconfigure window position
tli_lut_struct_para_init	initialize the parameters of TLI layer LUT structure with the default values, it is suggested that call this function after a tli_layer_lut_parameter_struct structure is defined
tli_lut_init	initialize TLI layer LUT
tli_color_key_init	initialize TLI layer color key

Function name	Function description
tli_layer_enable	enable TLI layer
tli_layer_disable	disable TLI layer
tli_color_key_enable	enable TLI layer color keying
tli_color_key_disable	disable TLI layer color keying
tli_lut_enable	enable TLI layer LUT
tli_lut_disable	disable TLI layer LUT
tli_line_mark_set	set line mark value
tli_current_pos_get	get current displayed position
tli_interrupt_enable	enable TLI interrupt
tli_interrupt_disable	disable TLI interrupt
tli_interrupt_flag_get	get TLI interrupt flag
tli_interrupt_flag_clear	clear TLI interrupt flag
tli_flag_get	get TLI flag or state in TLI_INTF register or TLI_STAT register

### Structure tli\_parameter\_struct

**Table3-851. Structure tli\_parameter\_struct**

Member name	Function description
synpsz_vpsz	size of the vertical synchronous pulse
synpsz_hpsz	size of the horizontal synchronous pulse
backpsz_vbpsz	size of the vertical back porch plus synchronous pulse
backpsz_hbpsz	size of the horizontal back porch plus synchronous pulse
activesz_vasz	size of the vertical active area width plus back porch and synchronous pulse
activesz_hasz	size of the horizontal active area width plus back porch and synchronous pulse
totalsz_vtsz	vertical total size of the display
totalsz_htsz	horizontal total size of the display
backcolor_red	background value red
backcolor_green	background value green

Member name	Function description
backcolor_blue	background value blue
signalpolarity_hs	horizontal pulse polarity selection
signalpolarity_vs	vertical pulse polarity selection
signalpolarity_de	data enable polarity selection
signalpolarity_pixelck	pixel clock polarity selection

### Structure tli\_layer\_parameter\_struct

**Table3-852. Structure tli\_layer\_parameter\_struct**

Member name	Function description
layer_window_rightpos	window right position
layer_window_leftpos	window left position
layer_window_bottompos	window bottom position
layer_window_toppos	window top position
layer_ppf	packeted pixel format
layer_sa	specified alpha
layer_default_alpha	the default color alpha
layer_default_red	the default color red
layer_default_green	the default color green
layer_default_blue	the default color blue
layer_acf1	alpha calculation factor 1 of blending method
layer_acf2	alpha calculation factor 2 of blending method
layer_frame_bufaddr	frame buffer base address
layer_frame_buf_stride_offset	frame buffer stride offset
layer_frame_line_length	frame line length
layer_frame_total_line_number	frame total line number

## Structure tli\_layer\_lut\_parameter\_struct

**Table3-853. Structure tli\_layer\_lut\_parameter\_struct**

Member name	Function description
layer_table_addr	look up table write address
layer_lut_channel_red	red channel of a LUT entry
layer_lut_channel_green	green channel of a LUT entry
layer_lut_channel_blue	blue channel of a LUT entry

## tli\_deinit

The description of tli\_deinit is shown as below:

**Table 3-854. Function tli\_deinit**

Function name	tli_deinit
Function prototype	void tli_deinit(void);
Function descriptions	deinitialize TLI registers
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize TLI */
tli_deinit();
```

## tli\_struct\_para\_init

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

**Table 3-855. Function tli\_struct\_para\_init**

Function name	tli_struct_para_init
---------------	----------------------

<b>Function prototype</b>	void tli_struct_para_init(tli_parameter_struct *tli_struct);
<b>Function descriptions</b>	initialize the parameters of TLI parameter structure with the default values, it is suggested that call this function after a tli_parameter_struct structure is defined
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>*tli_struct</b>	a pointer to tli_parameter_struct
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
tli_parameter_struct tli_init_struct;
```

```
/* initialize the parameters of TLI parameter structure with the default values */
```

```
tli_struct_para_init(&tli_init_struct);
```

## tli\_init

The description of tli\_init is shown as below:

**Table 3-856. Function tli\_init**

<b>Function name</b>	tli_init
<b>Function prototype</b>	void tli_init(tli_parameter_struct *tli_struct);
<b>Function descriptions</b>	initialize TLI display timing parameters
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>*tli_struct</b>	a pointer to tli_parameter_struct
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
tli_parameter_struct tli_init_struct;

/* initialize the parameters of TLI parameter structure with the default values */

tli_struct_para_init(&tli_init_struct);

/* configure TLI parameter struct */

tli_init_struct.signalpolarity_hs = TLI_HSYN_ACTLIVE_LOW;
tli_init_struct.signalpolarity_vs = TLI_VSYN_ACTLIVE_LOW;
tli_init_struct.signalpolarity_de = TLI_DE_ACTLIVE_LOW;
tli_init_struct.signalpolarity_pixelck = TLI_PIXEL_CLOCK_TLI;

/* LCD display timing configuration */

tli_init_struct.synpsz_hpsz = 40;
tli_init_struct.synpsz_vpsz = 9;
tli_init_struct.backpsz_hbpsz = 42;
tli_init_struct.backpsz_vbpsz = 11;
tli_init_struct.activesz_hasz = 522;
tli_init_struct.activesz_vasz = 283;
tli_init_struct.totalsz_htsz = 524;
tli_init_struct.totalsz_vtsz = 285;

/* configure LCD background R,G,B values */

tli_init_struct.backcolor_red = 0xFF;
tli_init_struct.backcolor_green = 0xFF;
tli_init_struct.backcolor_blue = 0xFF;

tli_init(&tli_init_struct);
```

### tli\_dither\_config

The description of tli\_dither\_config is shown as below:

**Table 3-857. Function tli\_dither\_config**

Function name	tli_dither_config
Function prototype	void tli_dither_config(uint8_t dither_stat);

<b>Function descriptions</b>	configure TLI dither function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dither_stat</b>	dither function
<i>TLI_DITHER_ENABLE</i>	enable TLI dither function
<i>TLI_DITHER_DISABLE</i>	disable TLI dither function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TLI dither function */
```

```
tli_dither_config(TLI_DITHER_ENABLE);
```

### tli\_enable

The description of tli\_enable is shown as below:

**Table 3-858. Function tli\_enable**

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



Example:

```
/* enabled TLI */
tli_enable();
```

### tli\_disable

The description of tli\_disable is shown as below:

**Table 3-859. Function tli\_disable**

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

Example:

```
/* disable TLI */
tli_disable();
```

### tli\_reload\_config

The description of tli\_reload\_config is shown as below:

**Table 3-860. Function tli\_reload\_config**

Function name	tli_reload_config
Function prototype	void tli_reload_config(uint8_t reload_mod);
Function descriptions	configure TLI reload mode
Precondition	-
The called functions	-

Input parameter{in}	
<b>reload_mod</b>	TLI reload mode
<i>TLI_FRAME_BLANK_RELOAD_EN</i>	the layer configuration will be reloaded at frame blank
<i>TLI_REQUEST_RELOAD_EN</i>	the layer configuration will be reloaded after this bit sets
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TLI reload mode */
```

```
tli_reload_config(TLI_REQUEST_RELOAD_EN);
```

### tli\_layer\_struct\_para\_init

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

**Table 3-861. Function tli\_layer\_struct\_para\_init**

<b>Function name</b>	tli_layer_struct_para_init
<b>Function prototype</b>	void tli_layer_struct_para_init(tli_layer_parameter_struct *layer_struct);
<b>Function descriptions</b>	initialize the parameters of TLI layer structure with the default values, it is suggested that call this function after a tli_layer_parameter_struct structure is defined
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>*layer_struct</b>	a pointer to tli_layer_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

tli_layer_parameter_struct tli_layer_init_struct;

/* initialize the parameters of TLI layer structure with the default values */

tli_layer_struct_para_init(&tli_layer_init_struct);

```

## tli\_layer\_init

The description of tli\_layer\_init is shown as below:

**Table 3-862. Function tli\_layer\_init**

<b>Function name</b>	tli_layer_init
<b>Function prototype</b>	void tli_layer_init(uint32_t layerx,tli_layer_parameter_struct *layer_struct);
<b>Function descriptions</b>	initialize TLI layer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>layerx</b>	layer
<i>LAYERx</i>	x=0, 1
<b>Input parameter{in}</b>	
<b>*layer_struct</b>	a pointer to tli_layer_parameter_struct
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

tli_layer_parameter_struct tli_layer_init_struct;

tli_layer_struct_para_init(&tli_layer_init_struct);

/* TLI layer0 configuration */

tli_layer_init_struct.layer_window_leftpos = 20 + 43;

tli_layer_init_struct.layer_window_rightpos = (20 + 440 + 43 - 1);

tli_layer_init_struct.layer_window_toppos = 40 + 12;

tli_layer_init_struct.layer_window_bottompos = (40 + 182 + 12 - 1);

tli_layer_init_struct.layer_ppf = LAYER_PPF_RGB565;

```

```

/* TLI window specified alpha configuration */

tli_layer_init_struct.layer_sa = 255;

/* TLI layer default alpha R,G,B value configuration */

tli_layer_init_struct.layer_default_blue = 0xFF;

tli_layer_init_struct.layer_default_green = 0xFF;

tli_layer_init_struct.layer_default_red = 0xFF;

tli_layer_init_struct.layer_default_alpha = 0xFF;

/* TLI window blend configuration */

tli_layer_init_struct.layer_acf1 = LAYER_ACF1_PASA;

tli_layer_init_struct.layer_acf2 = LAYER_ACF2_PASA;

/* TLI layer frame buffer base address configuration */

tli_layer_init_struct.layer_frame_bufaddr = (uint32_t)&gBackground;

tli_layer_init_struct.layer_frame_line_length = ((440 * 2) + 3);

tli_layer_init_struct.layer_frame_buf_stride_offset = (440 * 2);

tli_layer_init_struct.layer_frame_total_line_number = 182;

tli_layer_init(LAYER0, &tli_layer_init_struct);

```

### tli\_layer\_window\_offset\_modify

The description of tli\_layer\_window\_offset\_modify is shown as below:

**Table 3-863. Function tli\_layer\_window\_offset\_modify**

<b>Function name</b>	tli_layer_window_offset_modify
<b>Function prototype</b>	void tli_layer_window_offset_modify(uint32_t layerx,uint16_t offset_x,uint16_t offset_y);
<b>Function descriptions</b>	reconfigure window position
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>layerx</b>	layer
<b>LAYERx</b>	x=0, 1
<b>Input parameter{in}</b>	

<b>offset_x</b>	new horizontal offset
<b>Input parameter{in}</b>	
<b>offset_y</b>	new vertical offset
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reconfigure LAYER1 window position */
```

```
tli_layer_window_offset_modify(LAYER1, 20, 20);
```

### tli\_lut\_struct\_para\_init

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

**Table 3-864. Function tli\_lut\_struct\_para\_init**

<b>Function name</b>	tli_lut_struct_para_init
<b>Function prototype</b>	void tli_lut_struct_para_init(tli_layer_lut_parameter_struct *lut_struct);
<b>Function descriptions</b>	initialize the parameters of TLI layer LUT structure with the default values, it is suggested that call this function after a tli_layer_lut_parameter_struct structure is defined
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>*lut_struct</b>	a pointer to tli_layer_lut_parameter_struct
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
tli_layer_lut_parameter_struct tli_lut_struct;
```

```
/* initialize the parameters of TLI layer LUT structure with the default values */
```

```
tli_lut_struct_para_init(&tli_lut_struct);
```

## tli\_lut\_init

The description of tli\_lut\_init is shown as below:

**Table 3-865. Function tli\_lut\_init**

<b>Function name</b>	tli_lut_init
<b>Function prototype</b>	void tli_lut_init(uint32_t layerx,tli_layer_lut_parameter_struct *lut_struct);
<b>Function descriptions</b>	initialize TLI layer LUT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>layerx</b>	layer
<i>LAYERx</i>	x=0, 1
<b>Input parameter{in}</b>	
<b>*lut_struct</b>	a pointer to tli_layer_lut_parameter_struct
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
tli_layer_lut_parameter_struct tli_lut_struct;

tli_lut_struct_para_init(&tli_lut_struct);

/* initialize TLI layer0 LUT */

tli_lut_struct.layer_table_addr = 0x20003000;

tli_lut_struct.layer_lut_channel_red = 0x20;

tli_lut_struct.layer_lut_channel_green = 0x30;

tli_lut_struct.layer_lut_channel_blue = 0xFF;

tli_lut_init(LAYER0, &tli_lut_struct);
```

## tli\_color\_key\_init

The description of tli\_color\_key\_init is shown as below:

Table 3-866. Function tli\_color\_key\_init

Function name	tli_color_key_init
Function prototype	void tli_color_key_init(uint32_t layerx,uint32_t redkey,uint32_t greenkey,uint32_t bluekey);
Function descriptions	initialize TLI layer color key
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
LAYERx	x=0, 1
Input parameter{in}	
redkey	color key red
Input parameter{in}	
greenkey	color key green
Input parameter{in}	
bluekey	color key blue
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TLI layer0 color key */
tli_color_key_init(LAYER0, 0xAA, 0xFF, 0x00);
```

### tli\_layer\_enable

The description of tli\_layer\_enable is shown as below:

Table 3-867. Function tli\_layer\_enable

Function name	tli_layer_enable
Function prototype	void tli_layer_enable(uint32_t layerx);
Function descriptions	enable TLI layer

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>layerx</b>	layer
<i>LAYERx</i>	x=0, 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TLI layer0 */
```

```
tli_layer_enable(LAYER0);
```

### tli\_layer\_disable

The description of tli\_layer\_disable is shown as below:

**Table 3-868. Function tli\_layer\_disable**

<b>Function name</b>	tli_layer_disable
<b>Function prototype</b>	void tli_layer_disable(uint32_t layerx);
<b>Function descriptions</b>	disable TLI layer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>layerx</b>	layer
<i>LAYERx</i>	x=0, 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TLI layer0 */
```



```
tli_layer_disable(LAYER0);
```

### tli\_color\_key\_enable

The description of tli\_color\_key\_enable is shown as below:

**Table 3-869. Function tli\_color\_key\_enable**

<b>Function name</b>	tli_color_key_enable
<b>Function prototype</b>	void tli_color_key_enable(uint32_t layerx);
<b>Function descriptions</b>	enable TLI layer color keying
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>layerx</b>	layer
<i>LAYERx</i>	x=0, 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TLI layer0 color keying */
```

```
tli_color_key_enable(LAYER0);
```

### tli\_color\_key\_disable

The description of tli\_color\_key\_disable is shown as below:

**Table 3-870. Function tli\_color\_key\_disable**

<b>Function name</b>	tli_color_key_disable
<b>Function prototype</b>	void tli_color_key_disable(uint32_t layerx);
<b>Function descriptions</b>	disable TLI layer color keying
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>layerx</b>	layer
<i>LAYERx</i>	x=0, 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TLI layer0 color keying */
tli_color_key_disable(LAYER0);
```

### tli\_lut\_enable

The description of tli\_lut\_enable is shown as below:

**Table 3-871. Function tli\_lut\_enable**

<b>Function name</b>	tli_lut_enable
<b>Function prototype</b>	void tli_lut_enable(uint32_t layerx);
<b>Function descriptions</b>	enable TLI layer LUT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>layerx</b>	layer
<i>LAYERx</i>	x=0, 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TLI layer0 LUT */
tli_lut_enable(LAYER0);
```

## tli\_lut\_disable

The description of tli\_lut\_disable is shown as below:

**Table 3-872. Function tli\_lut\_disable**

<b>Function name</b>	tli_lut_disable
<b>Function prototype</b>	void tli_lut_disable(uint32_t layerx);
<b>Function descriptions</b>	disable TLI layer0 LUT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>layerx</b>	layer
<i>LAYERx</i>	x=0, 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TLI layer0 LUT */
tli_lut_disable(LAYER0);
```

## tli\_line\_mark\_set

The description of tli\_line\_mark\_set is shown as below:

**Table 3-873. Function tli\_line\_mark\_set**

<b>Function name</b>	tli_line_mark_set
<b>Function prototype</b>	void tli_line_mark_set(uint32_t line_num);
<b>Function descriptions</b>	set line mark value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>line_num</b>	line number
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

Example:

```
/* set line mark value */
```

```
tli_line_mark_set(0x20);
```

### tli\_current\_pos\_get

The description of tli\_current\_pos\_get is shown as below:

**Table 3-874. Function tli\_current\_pos\_get**

<b>Function name</b>	tli_current_pos_get
<b>Function prototype</b>	uint32_t tli_current_pos_get(void);
<b>Function descriptions</b>	get current displayed position
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	position of current pixel

Example:

```
uint32_t pos;
```

```
/* get current pixel position */
```

```
pos = tli_current_pos_get();
```

### tli\_interrupt\_enable

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

**Table 3-875. Function tli\_interrupt\_enable**

<b>Function name</b>	tli_interrupt_enable
----------------------	----------------------

<b>Function prototype</b>	void tli_interrupt_enable(uint32_t int_flag);
<b>Function descriptions</b>	enable TLI interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	TLI interrupt flags
<i>TLI_INT_LM</i>	line mark interrupt
<i>TLI_INT_FE</i>	FIFO error interrupt
<i>TLI_INT_TE</i>	transaction error interrupt
<i>TLI_INT_LCR</i>	layer configuration reloaded interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TLI line mark interrupt */
tli_interrupt_enable(TLI_INT_LM);
```

### tli\_interrupt\_disable

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

**Table 3-876. Function tli\_interrupt\_disable**

<b>Function name</b>	tli_interrupt_disable
<b>Function prototype</b>	void tli_interrupt_disable(uint32_t int_flag);
<b>Function descriptions</b>	disable TLI interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	TLI interrupt flags
<i>TLI_INT_LM</i>	line mark interrupt

<i>TLI_INT_FE</i>	FIFO error interrupt
<i>TLI_INT_TE</i>	transaction error interrupt
<i>TLI_INT_LCR</i>	layer configuration reloaded interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TLI line mark interrupt */
```

```
tli_interrupt_disable(TLI_INT_LM);
```

### tli\_interrupt\_flag\_get

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

**Table 3-877. Function tli\_interrupt\_flag\_get**

<b>Function name</b>	tli_interrupt_flag_get
<b>Function prototype</b>	FlagStatus tli_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	get TLI interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	TLI interrupt flags
<i>TLI_INT_FLAG_LM</i>	line mark interrupt flag
<i>TLI_INT_FLAG_FE</i>	FIFO error interrupt flag
<i>TLI_INT_FLAG_TE</i>	transaction error interrupt flag
<i>TLI_INT_FLAG_LCR</i>	layer configuration reloaded interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TLI interrupt flag */
if(SET == tli_interrupt_flag_get(TLI_INT_FLAG_LM)){
    tli_interrupt_flag_clear(TLI_INT_FLAG_LM);
}
```

### tli\_interrupt\_flag\_clear

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

**Table 3-878. Function tli\_interrupt\_flag\_clear**

<b>Function name</b>	tli_interrupt_flag_clear
<b>Function prototype</b>	void tli_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear TLI interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag</b>	TLI interrupt flags
<i>TLI_INT_FLAG_LM</i>	line mark interrupt flag
<i>TLI_INT_FLAG_FE</i>	FIFO error interrupt flag
<i>TLI_INT_FLAG_TE</i>	transaction error interrupt flag
<i>TLI_INT_FLAG_LCR</i>	layer configuration reloaded interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
if(SET == tli_interrupt_flag_get(TLI_INT_FLAG_LM)){
    /* clear TLI interrupt flag */
    tli_interrupt_flag_clear(TLI_INT_FLAG_LM);
}
```

## tli\_flag\_get

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

**Table 3-879. Function tli\_flag\_get**

<b>Function name</b>	tli_flag_get
<b>Function prototype</b>	FlagStatus tli_flag_get(uint32_t flag);
<b>Function descriptions</b>	get TLI flag or state in TLI_INTF register or TLI_STAT register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	TLI flags or states
<i>TLI_FLAG_VDE</i>	current VDE state
<i>TLI_FLAG_HDE</i>	current HDE state
<i>TLI_FLAG_VS</i>	current VS status of the TLI
<i>TLI_FLAG_HS</i>	current HS status of the TLI
<i>TLI_FLAG_LM</i>	line mark interrupt flag
<i>TLI_FLAG_FE</i>	FIFO error interrupt flag
<i>TLI_FLAG_TE</i>	transaction error interrupt flag
<i>TLI_FLAG_LCR</i>	layer configuration reloaded interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* wait the TLI_FLAG_LM flag set */
while(RESET == tli_flag_get(TLI_FLAG_LM)){
}
```

## 3.28. TRNG

The true random number generator (TRNG) can generate a 32-bit random value by using



continuous analog noise. The TRNG registers are listed in chapter [3.28.1](#), the TRNG firmware functions are introduced in chapter [3.28.2](#).

### 3.28.1. Descriptions of Peripheral registers

TRNG registers are listed in the table shown as below:

**Table 3-880 TRNG Registers**

Registers	Descriptions
TRNG_CTL	TRNG control register
TRNG_STAT	TRNG status register
TRNG_DATA	TRNG data register

### 3.28.2. Descriptions of Peripheral functions

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

**Table 3-881. TRNG firmware function**

Function name	Function description
trng_deinit	deinitialize the TRNG
trng_enable	enable the TRNG interface
trng_disable	disable the TRNG interface
trng_get_true_random_data	get the true random data
trng_interrupt_enable	enable the TRNG interrupt
trng_interrupt_disable	disable the TRNG interrupt
trng_flag_get	get the TRNG status flags
trng_interrupt_flag_get	get the TRNG interrupt flags
trng_interrupt_flag_clear	clear the TRNG interrupt flags

#### trng\_deinit

The description of trng\_deinit is shown as below:

**Table 3-882. Function trng\_deinit**

Function name	trng_deinit
Function prototype	void trng_deinit(void)
Function descriptions	reset TRNG peripheral

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

Example:

```
/* reset TRNG */
```

```
trng_deinit();
```

### trng\_enable

The description of trng\_enable is shown as below:

**Table 3-883. Function trng\_enable**

<b>Function name</b>	trng_enable
<b>Function prototype</b>	void trng_enable(void)
<b>Function descriptions</b>	enable the TRNG interface
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable TRNG */
```

```
trng_enable();
```

## trng\_disable

The description of trng\_disable is shown as below:

**Table 3-884 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 3-885 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>	
-	-

Return value	
uint32_t	the generated random data

Example:

```
/* get true random data */
uint32_t data;
data = trng_get_true_random_data();
```

### trng\_interrupt\_enable

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

**Table 3-886 Function trng\_interrupt\_enable**

Function name	trng_interrupt_enable
Function prototype	void trng_interrupt_enable(void)
Function descriptions	enable the TRNG interrupt
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TRNG interrupt */
trng_interrupt_enable();
```

### trng\_interrupt\_disable

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

**Table 3-887 Function trng\_interrupt\_disable**

Function name	trng_interrupt_disable
Function prototype	void trng_interrupt_disable(void)

<b>Function descriptions</b>	disable the TRNG interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TRNG interrupt */
```

```
trng_interrupt_disable();
```

### trng\_flag\_get

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

**Table 3-888 Function 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 status flag
<i>TRNG_FLAG_DRDY</i>	Random Data ready status
<i>TRNG_FLAG_CECS</i>	Clock error current status
<i>TRNG_FLAG_SECS</i>	Seed error current status
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

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

Example:

```
/* get TRNG clock error current flag status*/
```

```
FlagStatus flag_status = RESET;
```

```
flag_status == trng_flag_get(TRNG_FLAG_CECS);
```

### trng\_interrupt\_flag\_get

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

**Table 3-889 Function 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>int_flag</b>	trng interrupt flag
<i>TRNG_INT_FLAG_CEIF</i>	clock error interrupt flag
<i>TRNG_INT_FLAG_SEIF</i>	Seed error interrupt flag
<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 3-890 Function 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>int_flag</b>	trng interrupt flag
TRNG_INT_FLAG_CEIF	clock error interrupt flag
TRNG_INT_FLAG_SEIF	Seed error interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TRNG clock error interrupt flag*/
trng_interrupt_flag_clear(TRNG_INT_FLAG_CEIF);
```

## 3.29. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.29.1](#), the USART firmware functions are introduced in chapter [3.29.2](#).

### 3.29.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

**Table 3-891. USART Registers**

Registers	Descriptions
USART_STAT0	Status register 0
USART_DATA	Data register
USART_BAUD	Baud rate register

Registers	Descriptions
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

### 3.29.2. Descriptions of Peripheral functions

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

**Table 3-892. 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
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



Function name	Function description
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
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

## usart\_deinit

The description of usart\_deinit is shown as below:

**Table 3-893. 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=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset USART0 */
usart_deinit (USART0);
```

### usart\_baudrate\_set

The description of usart\_baudrate\_set is shown as below:

**Table 3-894. 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
Input parameter{in}	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
<b>baudval</b>	baud rate value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 baud rate value */
usart_baudrate_set(USART0, 115200);
```

### usart\_parity\_config

The description of usart\_parity\_config is shown as below:

**Table 3-895. 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=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<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 3-896. 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=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<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 3-897. 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=0,1,2,5
UARTx	x=3,4,6,7
<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=3,4,6,7)
USART_STB_2BIT	2 bits
USART_STB_1_5BIT	1.5 bits, not available for UARTx(x=3,4,6,7)
<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 3-898. 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>	-

The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 */
```

```
usart_enable(USART0);
```

### usart\_disable

The description of usart\_disable is shown as below:

**Table 3-899. Function usart\_disable**

Function name	usart_disable
Function prototype	void usart_disable(uint32_t usart_periph);
Function descriptions	disable USART
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 */
```

```
usart_disable(USART0);
```

### usart\_transmit\_config

The description of usart\_transmit\_config is shown as below:

**Table 3-900. Function usart\_transmit\_config**

Function name	usart_transmit_config
Function prototype	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=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<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 3-901. 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=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<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>	
-	-

Return value	
-	-

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 3-902. Function usart\_data\_first\_config**

Function name	usart_data_first_config
Function prototype	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
Function descriptions	data is transmitted/received with the LSB/MSB first
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Input parameter{in}	
msbf	LSB first or MSB first
USART_MSBF_LSB	LSB first
USART_MSBF_MSB	MSB first
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure LSB of data first */
```

```
usart_data_first_config(USART0, USART_MSBF_LSB);
```

### usart\_invert\_config

The description of usart\_invert\_config is shown as below:

**Table 3-903. Function usart\_invert\_config**

Function name	usart_invert_config
Function prototype	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
Function descriptions	configure USART inversion
Precondition	-
The called functions	-

Input parameter{in}	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
invertpara	refer to enum usart_invert_enum
<i>USART_DINV_ENAB LE</i>	data bit level inversion
<i>USART_DINV_DISAB LE</i>	data bit level not inversion
<i>USART_TXPIN_ENAB LE</i>	TX pin level inversion
<i>USART_TXPIN_DISAB LE</i>	TX pin level not inversion
<i>USART_RXPIN_ENAB LE</i>	RX pin level inversion
<i>USART_RXPIN_DISAB LE</i>	RX pin level not inversion
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 3-904. Function usart\_oversample\_config**

<b>Function name</b>	usart_oversample_config
<b>Function prototype</b>	void usart_oversample_config(uint32_t usart_periph, uint32_t oversamp);
<b>Function descriptions</b>	configure the USART oversample mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
<b>oversamp</b>	oversample value
<i>USART_OVSMOD_8</i>	8 bits
<i>USART_OVSMOD_16</i>	16 bits



Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 oversample mode */
usart_oversample_config(USART0, USART_OVSMOD_8);
```

### usart\_sample\_bit\_config

The description of usart\_sample\_bit\_config is shown as below:

**Table 3-905. Function usart\_sample\_bit\_config**

Function name	usart_sample_bit_config
Function prototype	void usart_sample_bit_config(uint32_t usart_periph, uint32_t obsm);
Function descriptions	configure sample bit method
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
obsm	sample bit
USART_OSB_1bit	1 bits
USART_OSB_3bit	3 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 sample bit */
usart_sample_bit_config(USART0, USART_OSB_1bit);
```

### usart\_receiver\_timeout\_enable

The description of usart\_receiver\_timeout\_enable is shown as below:

**Table 3-906. Function usart\_receiver\_timeout\_enable**

Function name	usart_receiver_timeout_enable
Function prototype	void usart_receiver_timeout_enable(uint32_t usart_periph);
Function descriptions	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=0,1,2,5
<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 3-907. 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=0,1,2,5
<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 3-908. Function usart\_receiver\_timeout\_threshold\_config**

<b>Function name</b>	usart_receiver_timeout_threshold_config
<b>Function prototype</b>	void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);

<b>Function descriptions</b>	configure receiver timeout threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<b>Input parameter{in}</b>	
<b>rtimeout</b>	timeout value
<i>0-0xFFFFF</i>	timeout value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the receiver timeout threshold of USART0 */
```

```
usart_receiver_timeout_threshold_config(USART0, 115200*3);
```

### usart\_data\_transmit

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

**Table 3-909. Function usart\_data\_transmit**

<b>Function name</b>	usart_data_transmit
<b>Function prototype</b>	void usart_data_transmit(uint32_t usart_periph, uint32_t data);
<b>Function descriptions</b>	USART transmit data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>Input parameter{in}</b>	
<b>data</b>	data of transmission
<i>0-0x1FF</i>	data of transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 transmit data */
```

```
usart_data_transmit(USART0, 0xAA);
```

## usart\_data\_receive

The description of usart\_data\_receive is shown as below:

**Table 3-910. Function usart\_data\_receive**

<b>Function name</b>	usart_data_receive
<b>Function prototype</b>	void usart_receive_config(uint32_t usart_periph);
<b>Function descriptions</b>	USART receive data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	data of received(0-0x1FF)

Example:

```
/* USART0 receive data */

uint16_t temp;

temp = usart_data_receive(USART0);
```

## usart\_address\_config

The description of usart\_address\_config is shown as below:

**Table 3-911. Function usart\_address\_config**

<b>Function name</b>	usart_address_config
<b>Function prototype</b>	void usart_address_config(uint32_t usart_periph, uint8_t addr);
<b>Function descriptions</b>	configure the address of the USART in wake up by address match mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>Input parameter{in}</b>	
<b>addr</b>	address of USART/UART
<i>0-0xFF</i>	address of USART/UART
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

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 3-912. 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
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
<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 3-913. 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
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
<b>Output parameter{out}</b>	

-	-
Return value	
-	-

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 3-914. 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
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>Input parameter{in}</b>	
<b>wmethod</b>	two methods be used to enter or exit the mute mode
<i>USART_WM_IDLE</i>	idle line
<i>USART_WM_ADDR</i>	address 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 3-915. 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
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 LIN mode enable */
```

```
usart_lin_mode_enable(USART0);
```

### usart\_lin\_mode\_disable

The description of usart\_lin\_mode\_disable is shown as below:

**Table 3-916. Function usart\_lin\_mode\_disable**

<b>Function name</b>	usart_lin_mode_disable
<b>Function prototype</b>	void usart_lin_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable LIN mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 LIN mode disable */
```

```
usart_lin_mode_disable(USART0);
```

### usart\_lin\_break\_dection\_length\_config

The description of usart\_lin\_break\_dection\_length\_config is shown as below:

**Table 3-917. Function usart\_lin\_break\_dection\_length\_config**

<b>Function name</b>	usart_lin_break_dection_length_config
----------------------	---------------------------------------

<b>Function prototype</b>	void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lblen);
<b>Function descriptions</b>	configure LIN break frame length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>Input parameter{in}</b>	
<b>lblen</b>	two methods be used to enter or exit the mute mode
<i>USART_LBLEN_10B</i>	break frame length is 10 bits
<i>USART_LBLEN_11B</i>	break frame length is 11 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LIN break frame length */
```

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

### usart\_send\_break

The description of usart\_send\_break is shown as below:

**Table 3-918. Function usart\_send\_break**

<b>Function name</b>	usart_send_break
<b>Function prototype</b>	void usart_send_break(uint32_t usart_periph);
<b>Function descriptions</b>	send break frame
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 send break frame */
```



```
usart_send_break(USART0);
```

### usart\_halfduplex\_enable

The description of usart\_halfduplex\_enable is shown as below:

**Table 3-919. 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=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<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 3-920. 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=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<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 3-921. 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=0,1,2,5
<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 3-922. 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=0,1,2,5
<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 3-923. 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=0,1,2,5
<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 3-924. Function `usart_guard_time_config`**

<b>Function name</b>	<code>usart_guard_time_config</code>
<b>Function prototype</b>	<code>void usart_guard_time_config(uint32_t usart_periph,uint32_t gaut);</code>
<b>Function descriptions</b>	configure guard time value in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<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 3-925. Function `usart_smartcard_mode_enable`**

<b>Function name</b>	<code>usart_smartcard_mode_enable</code>
<b>Function prototype</b>	<code>void usart_smartcard_mode_enable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	enable smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<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 3-926. 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=0,1,2,5
<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 3-927. 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=0,1,2,5
<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 3-928. 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=0,1,2,5
<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 3-929. 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=0,1,2,5
<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 3-930. 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=0,1,2,5
<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 3-931. 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=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

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 3-932. Function usart\_irda\_mode\_disable**

Function name	usart_irda_mode_disable
Function prototype	void usart_irda_mode_disable(uint32_t usart_periph);
Function descriptions	disable IrDA mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 IrDA mode */
```

```
usart_irda_mode_disable(USART0);
```

### usart\_prescaler\_config

The description of usart\_prescaler\_config is shown as below:

**Table 3-933. Function usart\_prescaler\_config**

Function name	usart_prescaler_config
Function prototype	void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);
Function descriptions	configure the peripheral clock prescaler in USART IrDA low-power mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7



Input parameter{in}	
<b>psc</b>	0x00-0xFF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the USART0 peripheral clock prescaler */
```

```
usart_prescaler_config(USART0, 0x00);
```

### usart\_irda\_lowpower\_config

The description of usart\_irda\_lowpower\_config is shown as below:

**Table 3-934. Function usart\_irda\_lowpower\_config**

<b>Function name</b>	usart_irda_lowpower_config
<b>Function prototype</b>	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
<b>Function descriptions</b>	configure IrDA low-power
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
<b>irlp</b>	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 3-935. Function usart\_hardware\_flow\_rts\_config**

<b>Function name</b>	usart_hardware_flow_rts_config
<b>Function prototype</b>	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
<b>Function descriptions</b>	configure hardware flow control RTS
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<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 3-936. 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=0,1,2,5
<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>	

-	-
Return value	
-	-

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 3-937. 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=0,1,2,5
<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 3-938. 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=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<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_word_length_set(USART0, USART_PCM_NONE);
```

### usart\_hardware\_flow\_coherence\_config

The description of usart\_hardware\_flow\_coherence\_config is shown as below:

**Table 3-939. 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=0,1,2,5
<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 3-940. Function usart\_dma\_receive\_config**

<b>Function name</b>	usart_dma_receive_config
<b>Function prototype</b>	void usart_dma_receive_config(uint32_t usart_periph, uint32_t dmacmd);
<b>Function descriptions</b>	configure USART DMA reception
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
<b>Input parameter{in}</b>	
<b>dmacmd</b>	enable or disable DMA for reception
USART_RECEIVE_DMA_ENABLE	DMA enable for reception
USART_RECEIVE_DMA_DISABLE	DMA disable for reception
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 DMA enable for reception */
```

```
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

### usart\_dma\_transmit\_config

The description of usart\_dma\_transmit\_config is shown as below:

**Table 3-941. Function usart\_dma\_transmit\_config**

<b>Function name</b>	usart_dma_transmit_config
<b>Function prototype</b>	void usart_dma_transmit_config(uint32_t usart_periph, uint32_t dmacmd);
<b>Function descriptions</b>	configure USART DMA transmission
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7

Input parameter{in}	
<b>dmacmd</b>	enable or disable DMA for transmission
<i>USART_TRANSMIT_DMA_ENABLE</i>	DMA enable for transmission
<i>USART_TRANSMIT_DMA_DISABLE</i>	DMA disable for transmission
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 DMA enable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

### usart\_flag\_get

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

**Table 3-942. 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=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
<b>flag</b>	USART flags, refer to usart_flag_enum
<i>USART_FLAG_CTS</i>	CTS change flag
<i>USART_FLAG_LBD</i>	LIN break detected flag
<i>USART_FLAG_TBE</i>	transmit data buffer empty flag
<i>USART_FLAG_TC</i>	transmission complete flag
<i>USART_FLAG_RBNE</i>	read data buffer not empty flag
<i>USART_FLAG_IDLE</i>	IDLE frame detected flag
<i>USART_FLAG_ORER</i> <i>R</i>	overrun error flag
<i>USART_FLAG_NERR</i>	noise error flag
<i>USART_FLAG_FERR</i>	frame error flag
<i>USART_FLAG_PERR</i>	parity error flag
<i>USART_FLAG_BSY</i>	busy flag
<i>USART_FLAG_EB</i>	end of block flag

<i>USART_FLAG_RT</i>	receiver timeout flag
<i>USART_FLAG_EPERR</i>	early parity error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get flag USART0 state */
```

```
FlagStatus status;
```

```
status = usart_flag_get(USART0,USART_FLAG_TBE);
```

### usart\_flag\_clear

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

**Table 3-943. 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=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to usart_flag_enum
<i>USART_FLAG_CTS</i>	CTS change flag
<i>USART_FLAG_LBD</i>	LIN break detected flag
<i>USART_FLAG_TC</i>	transmission complete
<i>USART_FLAG_RBNE</i>	read data buffer not empty
<i>USART_FLAG_EB</i>	end of block flag
<i>USART_FLAG_RT</i>	receiver timeout flag
<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 3-944. 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
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
<b>Input parameter{in}</b>	
<b>interrupt</b>	USART interrupt
USART_INT_PERR	parity error interrupt
USART_INT_TBE	transmitter buffer empty interrupt
USART_INT_TC	transmission complete interrupt
USART_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt
USART_INT_IDLE	IDLE line detected interrupt
USART_INT_LBD	LIN break detected interrupt
USART_INT_ERR	error interrupt
USART_INT_CTS	CTS interrupt
USART_INT_RT	receive timeout event interrupt
USART_INT_EB	end of block event interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 TBE interrupt */
```

```
usart_interrupt_enable(USART0, USART_INT_TBE);
```

## usart\_interrupt\_disable

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

**Table 3-945. 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=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>Input parameter{in}</b>	
<b>interrupt</b>	USART interrupt flag
<i>USART_INT_PERR</i>	parity error interrupt
<i>USART_INT_TBE</i>	transmitter buffer empty interrupt
<i>USART_INT_TC</i>	transmission complete interrupt
<i>USART_INT_RBNE</i>	read data buffer not empty interrupt and overrun error interrupt
<i>USART_INT_IDLE</i>	IDLE line detected interrupt
<i>USART_INT_LBD</i>	LIN break detected interrupt
<i>USART_INT_ERR</i>	error interrupt
<i>USART_INT_CTS</i>	CTS interrupt
<i>USART_INT_RT</i>	receive timeout event interrupt
<i>USART_INT_EB</i>	end of block event interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 TBE interrupt */
```

```
usart_interrupt_disable(USART0, USART_INT_TBE);
```

### usart\_interrupt\_flag\_get

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

**Table 3-946. 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=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>Input parameter{in}</b>	

int_flag	USART interrupt flag, refer to usart_interrupt_flag_enum
USART_INT_FLAG_PERRR	parity error interrupt and flag
USART_INT_FLAG_TBRE	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_ERORERR	error interrupt and overrun error
USART_INT_FLAG_EROR_NERR	error interrupt and noise error flag
USART_INT_FLAG_EROR_FERR	error interrupt and frame error flag
USART_INT_FLAG_EB	end of block event interrupt flag
USART_INT_FLAG_RT	receive timeout event interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the USART0 interrupt flag status */
```

```
FlagStatus status;
```

```
status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
```

### usart\_interrupt\_flag\_clear

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

**Table 3-947. Function usart\_interrupt\_flag\_clear**

Function name	usart_interrupt_flag_clear
Function prototype	void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
Function descriptions	clear USART interrupt flag 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=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag, refer to usart_interrupt_flag_enum
<i>USART_INT_FLAG_CTS</i>	CTS change flag
<i>USART_INT_FLAG_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>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the USART0 interrupt enable bit status */
```

```
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

## 3.30. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.30.1](#), the FWDGT firmware functions are introduced in chapter [3.30.2](#).

### 3.30.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

**Table 3-948. WWDGT Registers**

Registers	Descriptions
WWDGT_CTL	Control register
WWDGT_CFG	Configuration register
WWDGT_STAT	Status register

### 3.30.2. Descriptions of Peripheral functions

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

**Table 3-949. WWDGT firmware function**

Function name	Function description
wwdgt_deinit	reset the window watchdog timer configuration
wwdgt_enable	start the window watchdog timer counter
wwdgt_counter_update	configure the window watchdog timer counter value
wwdgt_config	configure counter value, window value, and prescaler divider value
wwdgt_interrupt_enable	enable early wakeup interrupt of WWDGT
wwdgt_flag_get	check early wakeup interrupt state of WWDGT
wwdgt_flag_clear	clear early wakeup interrupt state of WWDGT

#### wwdgt\_deinit

The description of wwdgt\_deinit is shown as below:

**Table 3-950. Function wwdgt\_deinit**

Function name	wwdgt_deinit
Function prototype	void wwdgt_deinit(void);
Function descriptions	reset the window watchdog timer configuration
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the window watchdog timer configuration */
```

```
wwdgt_deinit ( );
```

## wwdgt\_enable

The description of wwdgt\_enable is shown as below:

**Table 3-951. 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 3-952. Function wwdgt\_counter\_update**

<b>Function name</b>	wwdgt_counter_update
<b>Function prototype</b>	void wwdgt_counter_update(uint16_t counter_value);
<b>Function descriptions</b>	configure the window watchdog timer counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
counter_value	0x00 - 0x7F
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* update WWDGT counter to 0x7F */
```

```
wwdgt_counter_update(127);
```

### wwdgt\_config

The description of wwdgt\_config is shown as below:

**Table 3-953. Function wwdgt\_config**

Function name	wwdgt_config
Function prototype	void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);
Function descriptions	configure counter value, window value, and prescaler divider value
Precondition	-
The called functions	-
Input parameter{in}	
counter	0x00 - 0x7F
Input parameter{in}	
window	0x00 - 0x7F
Input parameter{in}	
prescaler	wwdgt prescaler value
WWDGT_CFG_PSC_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
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

### wwdgt\_interrupt\_enable

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

**Table 3-954. Function wwdgt\_interrupt\_enable**

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

Example:

```
/* enable early wakeup interrupt of WWDGT */
```

```
wwdgt_interrupt_enable ( );
```

### wwdgt\_flag\_get

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

**Table 3-955. Function wwdgt\_flag\_get**

<b>Function name</b>	wwdgt_flag_get
<b>Function prototype</b>	FlagStatus wwdgt_flag_get(void);
<b>Function descriptions</b>	check early wakeup interrupt state of WWDGT

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

Example:

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get ( );
```

```
if(status == RESET)
```

```
{
```

```
...
```

```
}else
```

```
{
```

```
...
```

```
}
```

### wwdgt\_flag\_clear

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

**Table 3-956. Function wwdgt\_flag\_clear**

<b>Function name</b>	wwdgt_flag_clear
<b>Function prototype</b>	void wwdgt_flag_clear(void);
<b>Function descriptions</b>	clear early wakeup interrupt state of WWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-



Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear early wakeup interrupt state of WWDGT */
```

```
wwdgt_flag_clear( );
```

## 4. Revision history

**Table 4-1. Revision history**

Revision No.	Description	Date
1.0	Initial Release	May.15, 2023
1.1	Update the Important Notice.	May.13, 2025

## 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, aeronautic or aerospace applications, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); and/or (iii) 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.

While the Company has implemented advanced security features, the Product may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on Customer's applications and products, and to the maximum extent permitted by applicable law, the Company accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

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.